OPTIMAL PARTITIONS FOR THE FAST MULTIPOLE METHOD

A Thesis
Presented to
The Academic Faculty

By

Lok Shan Wong

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Electrical and Computer Engineering

Georgia Institute of Technology

December 2016

OPTIMAL PARTITIONS FOR THE FAST MULTIPOLE METHOD

Approved by:

Dr. Christopher Barnes, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Justin Romberg
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Aaron Lanterman
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: December 8, 2016

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Christopher Barnes, for his time, guidance, and insight. His help has been crucial in my graduate studies and my research work.

I would also like to thank the others of my committee, Dr. Justin Romberg and Dr. Aaron Lanterman, for their time and input.

I would also like to thank Sandia National Laboratories for providing full financial support in my pursuit of a Master of Science degree. This opportunity allowed me to focus on my studies without having to worry about funding.

Finally, I would like to thank my parents for their love and support throughout my life. I cannot imagine being where I am today without them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

$\tilde{A}_2$	Coxeter system related to the triangular tessellation
$\tilde{A}_3$	Coxeter system related to the tetrahedral-octahedral honeycomb
l_n	Level-n
$\mathbb{N}^+$	Set of all positive integers
Q_n	The compactness of an n-dimensional simple rectilinear polytope
$\mathbb{R}^n$	n-dimensional space over the field of real numbers
Λ_n	Lattice isomorphic to the centers of n-dimensional FMM cells
FMM	Fast Multipole Method
SOP	Simple orthogonal polyhedron
SRP	Simple rectilinear polygon

SUMMARY

The fast multipole method (FMM) is an algorithm first developed to approximately solve the N-body problem in linear time. Part of the FMM involves recursively partitioning a region of source points into cells. The FMM normally uses a quadtree or octree structure, depending on whether the problem being solved is in 2D or 3D space, to partition this region.

Insight from studying lattices and covering problems leads to new, more efficient partitions for the FMM. New partitions are designed for 2D and 3D N-body problems based on the premise that more efficient partitioning results in fewer near-field and far-field calculations. These new partitions are tested against the standard partitions in FMM using computation time and relative error as metrics.

Results from 2D Monte Carlo simulations show little computation time reduction and significantly higher relative error. However, results from 3D Monte Carlo simulations show significant computation time reduction with little to no additional error in many cases.

CHAPTER 1

INTRODUCTION

In physics, the N-body problem seeks to determine how N bodies in space individually react to each other from each body's gravitational force. A similar N-body problem exists for determining how N charges individually react to each other from each charge's electrostatic force. A direct evaluation requires a matrix-vector product, which takes $\mathcal{O}(N^2)$ time. Solving an N-body problem quickly becomes prohibitive as N increases.

In 1987, Greengard and Roklin created the fast multipole method as a fast algorithm for solving the N-body problem [1]. The fast multipole method, or FMM, was designed to give approximate solutions to the N-body problem in $\mathcal{O}(N)$ time. The FMM has caused such a large impact since its inception, that the Society for Industrial and Applied Mathematics named the FMM one of the top 10 algorithms of the 20th century [2].

One component of the FMM is that the FMM recursively partitions a region of source points into cells. The FMM normally uses a quadtree structure for 2D problems and an octree structure for 3D problems. An important factor into how fast the FMM runs is the number of neighboring cells each cell in a partition has. This is because the number of neighboring cells affects the number of near-field and far-field calculations computed in the FMM. If one finds a partition of the region of sources but the number of neighboring cells each cell in the partition has decreases, then computation time decreases as well.

The goals of this thesis are to design, implement, and test new and efficient partitions for the FMM. We want partitions that will reduce the computation time in the FMM while not increasing approximation error in the FMM. Performance of these new partitions will be tested using open-source FMM programs as reference points. Performance will be evaluated using computation time and relative error as metrics.

The remainder of this thesis is organized as follows: Chapter 2 gives an overview of how the FMM works and previous research into different partitions and their effects. Chapter

3 gives an analysis on how partitions generally affect FMM performance and how one can find a partition that can minimize the number of neighboring cells of each cell in the partition. Chapter 4 develops a new partition in 2D FMM, describes how to implement this partition, and describes how the partition is tested. Chapter 5 describes content for 3D FMM similar to the content for 2D FMM in Chapter 4. Chapter 6 gives the conclusion of this thesis.

CHAPTER 2

BACKGROUND

This chapter provides background information about the fast multipole method and summarizes previous research into attempts to improve performance of the FMM. The first section is an overview about how the fast multipole method works. The second section summarizes past research into the effects of modifications to the standard FMM. The third section defines the problem statement that this thesis will address.

2.1 The Fast Multipole Method

The fast multipole method was first developed by Greengard and Rokhlin in 1987 to compute an approximate solution to the N-body problem in $O(N)$ or $O(N \log N)$ time [1, 3]. The FMM at its core is a fast algorithm to evaluate

$$f(y_j) = \sum_{i=1}^{N} K(y_j, x_i) w(x_i) \tag{2.1}$$

where y_j is the jth of M target points, x_i is the ith of N source points, $K(y_j, x_i)$ is a kernel function between x_i and y_j, and $w(x_i)$ is a scalar weight corresponding to source point x_i. A direct evaluation requires $O(MN)$ operations in the general case, or $O(N^2)$ operations in the N-body problem where the source points are also the target points. For ease of discussion, the FMM will be described in the context of the N-body problem. For future reference, the "standard" FMM will refer to the FMM as described in this section.

If the kernel function can be expressed as a low-rank approximation of the form

$$K(y_j, x_i) \approx \sum_{k=1}^{p} \phi_k(x_i) \psi_k(y_j) \tag{2.2}$$

then intermediate coefficients a_k can first be computed by

$$a_k = \sum_{i=1}^{N} w(x_i)\phi_k(x_i) \tag{2.3}$$

to be used to compute

$$f(y_j) \approx \sum_{k=1}^{p} a_k \psi_k(y_j). \tag{2.4}$$

Since the coefficients a_k only need to be computed once, evaluating $f(y_j)$ only costs $O(Np)$ operations. In general, $p \ll N$, so this computation is $O(N)$. This fact leads to the use of series expansions truncated after p terms to approximate $f(y_j)$.

The FMM as described by Greengard and Rokhlin [1] can be described in three stages. The first stage is initialization. The region of source points, or *problem region*, is encompassed in a square or a cube, called the *root cell*. The root cell is defined to be at level 0. The root cell is divided into disjoint *child cells* that each cover an equal amount of space in level 1. Each child cell is then recursively divided into finer-sized cells in a similar fashion until a criterion is reached. For example, the criterion may be to stop dividing if the cell has less than or equal to some arbitrary constant number of sources. A cell with child cells is a *parent cell*. A cell without a child cell is called a *leaf cell*. This process can be described as building a tree data structure. In 2D, each parent cell gets divided into four child cells, resulting in a *quadtree*. In 3D, each parent cell gets divided into eight child cells, resulting in an *octree*.

The second stage is the upward pass. Each leaf cell is represented by a *multipole expansion* computed from applying a particle-to-multipole (P2M) translation operator to each of the sources in the leaf cell. A cell's multipole expansion allows sources in the cell to be treated as if there is a single source, or *multipole*, at the center of the cell. Each parent cell in levels 1 and higher then gets a multipole expansion from applying a multipole-to-multipole (M2M) translation operator to each child cell's multipole.

The third stage is the downward pass. In each level from level 2 to the level with the

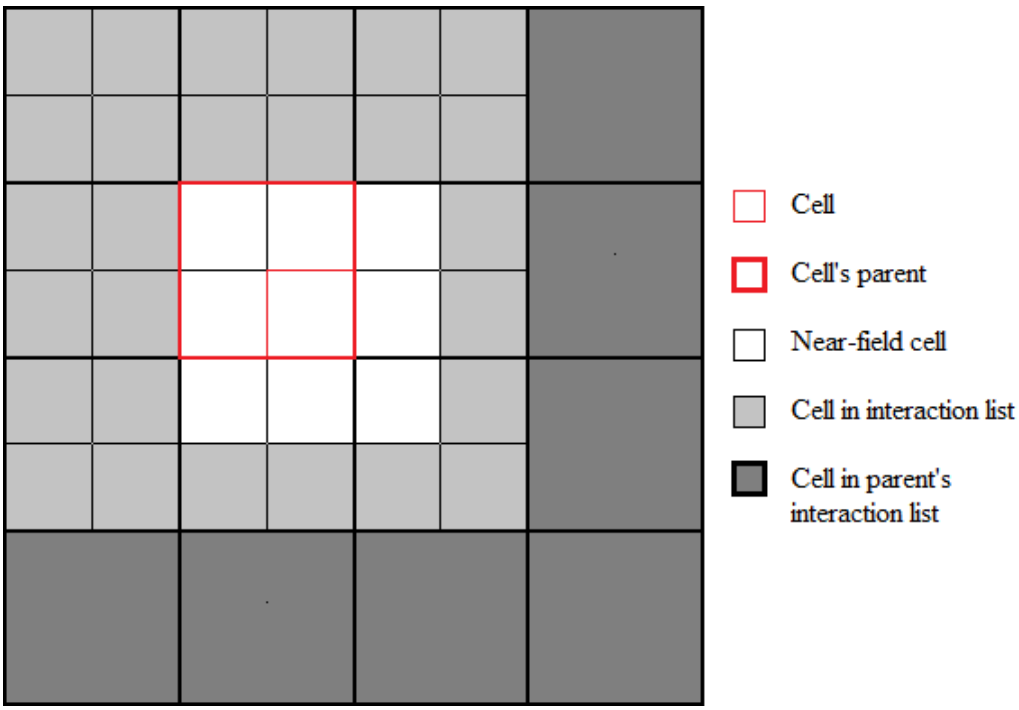
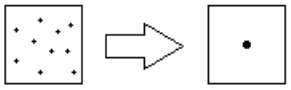
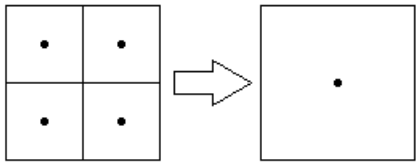


Figure 1: Example of the interaction list of one cell.

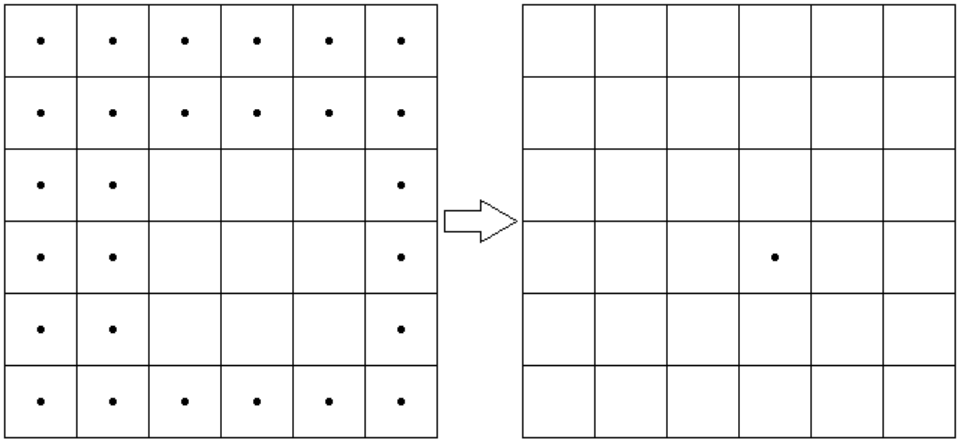
finest cells, every cell gets a *local expansion* by first applying a multipole-to-local (M2L) translation operator to the multipole of each cell in an *interaction list*. The interaction list of a cell consists of all cells in the same level that do not include the cell itself, cells that are adjecent, or *neighbors*, of the cell, and the children of cells in the parent cell's interaction list. In other words, the interaction list consists of far-field cells that had not already been accounted for in the parent cell's local expansion. Figure 1 gives a 2D example of the interaction list of one cell. The M2L operations for a cell gives the cell's multipole the approximate potential from all the sources in the far field. Each child cell then applies a local-to-local (L2L) translation operator to the parent's local expansion and uses the result as the cell's local expansion. Each leaf cell then applies a local-to-particle (L2P) translation operator from its local expansion to each target point in the cell and adds contributions from source points in the cell and in the cell's neighbors using a direct, or particle-to-particle (P2P), calculation. A direct calculation is necessary because approximative solutions break down in the near field. Figure 2 shows an example of each operation.



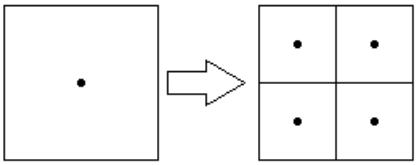
(a) P2M operations on points in cell to get multipole.



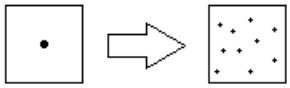
(b) M2M operations on child multipoles to get parent multipole.



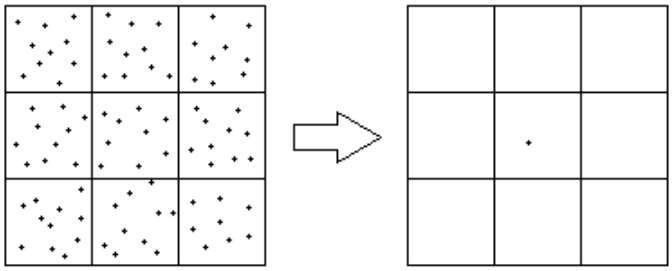
(c) M2L operations on multipoles in a cell's interaction list to get potential for a cell's multipole.



(d) L2L operation on parent multipole to get potential for each child multipole.



(e) L2P operation on multipole to get potential for each point.



(f) P2P operations on points in near field to get potential for target point.

Figure 2: 3D l_n cells for $n \leq 3$.

A key factor that allows the FMM to work is that the way near-field and far-field cells are defined allows for good error bounding [1]. Kernel functions and series expansions truncated after a few terms typically depend on the distance between two points. For a given number of terms used in expansions, the larger the distance between two points, the lower the error bound is between a direct calculation and an approximate calculation. Greengard and Rokhlin found that points at least 1.5 times the side length of a cell away from the center of the cell being operated on are far enough for approximate solutions to be arbitrarily close to actual solutions depending on the approximation method and the number of terms.

2.2 Modifications to the Standard FMM

There has been consideration of alternate partitions outside of quadtrees of square cells and octrees of cubic cells to improve performance of the FMM. Kudin and Scuseria [4] looked at using rectangular non-cubic cells in cases where the problem region and the distribution of points make the octree suboptimal. For example, the problem region might be much longer in one dimension. They found that the number of rectangular cells should be minimized to keep computation time low while getting higher accuracy in these cases. Anderson [5] proved that N-body algorithms that used spatially-balanced trees have a lower worst-case complexity than N-body algorithms that used density-balanced trees, and that triangular partitioning is inferior heuristically to the quadtree partitioning. Marzouk and Ghoniem [6] proposed using a k-means algorithm to first divide the problem region into clusters, and then using an octree for each cluster. The idea behind using k-means clustering was to create level 1 subregions that were more spherical and to solve the problem in a more parallelizable manner. This algorithm lowered computation time by an order of magnitude with a small increase in error compared to using the standard octree structure for the whole problem region. Marzouk and Ghoniem also found that allowing cells to overlap (i.e. sources can be in multiple cells) and using cells that are significantly longer in one dimension than in other dimensions are both result in higher errors.

All the partitions proposed in these papers have in most levels the same maximum number of neighbors a cell can have and the maximum number of cells in a cell's interaction list. That is, each cell has at most 8 (3^2-1) neighbors and 27 (6^2-3^2) cells in its interaction list in 2D, such as the example shown in Figure 1, and at most 26 (3^3-1) neighbors and 189 (6^3-3^3) cells in its interaction list in 3D.

2.3 Problem Statement

This thesis seeks a novel way to decrease computation time in the FMM without increasing error for a broad range of cases. The approach that will be taken is to design a new way of partitioning the problem domain that will reduce the number of near-field and far-field calculations. Solutions will be presented for the 2D and 3D cases. This work will first set up a theoretical foundation for the proposed solutions. Results from performance tests between open-source FMM codes and modified versions of the codes with the solutions implemented will demonstrate that the proposed modifications reduce computation time without sacrificing accuracy in many cases.

CHAPTER 3

THEORY

This chapter lays the theoretical foundation for the development of an algorithm designed to improve the FMM. The first section is an analysis of how different ways of partitioning the problem region affects the number of computations performed. The second section summarizes group theory topics relevant to this thesis. The third section connects the FMM and group theory together as a lead toward new solutions.

3.1 Effects of Partitioning on FMM Performance

This section is an original analysis into how standard partitioning in the FMM affects computation time. The goal is to explore potential ways computation savings can be achieved by adjusting the partition.

3.1.1 Upper Bound to Number of FMM Operations

As explained in the previous chapter, there are six types of operations that are used in FMM: P2M, M2M, M2L, L2L, L2P, and P2P. The P2M and L2P operations are between sources and leaf cells and only depend on the number of sources. These two operations apply to each source just once, regardless of how the sources are partitioned. The other four operations depend on how the sources are partitioned. M2M, M2L, and L2L are cell-to-cell operations, making the total number of these operations computed in the FMM inherently dependent on the number of cells in the partition. The number of P2P operations is the number of near-field operations. This is dependent on the partition in that the partition determines what is the near field and the far field of a cell. Therefore, a more *efficient partition* results in fewer partition-dependent operations.

Because determining the exact number of partition-dependent operations computed in the FMM would require knowing the source distribution, we seek an upper bound to the number of these operations. Consider an upper bound that is a function of four variables:

the number of levels l in the tree structure of cells, the maximum number of sources s each leaf cell can contain, the maximum number of child cells c each parent cell can be divided to, and the maximum number of neighbors n each cell can have, where $l, s, c, n \in \mathbb{N}^+$ and $c \geq 2$. Let B be an upper bound on the number of partition-dependent operations and B_x be an upper bound on the number of operations of type x. Then, the following equation holds true:

$$B = B_{M2M} + B_{M2L} + B_{L2L} + B_{P2P}. \tag{3.1}$$

M2M is an operation from a child cell to the child cell's parent that is computed once for each child cell of each parent cell in the tree. This means

$$B_{M2M} = \sum_{i=1}^{l-1} c^i = \frac{c^l - c}{c - 1}. \tag{3.2}$$

B_{M2M} is equal to the number of cells a tree with l levels and c branches out of each node in levels 0 to $l - 2$ (level $l - 1$ contains only leaf nodes) minus the root cell.

L2L is similar to M2M, except L2L is an operation from a parent cell to each of the parent's child cells. This means

$$B_{L2L} = B_{M2M} = \frac{c^l - c}{c - 1}. \tag{3.3}$$

B_{M2L} depends on the size of the interaction list of each cell. Recall from the previous chapter that the interaction list of a cell contains all cells in the same level that do not include the cell itself, the cell's neighbors, and the children of cells in the parent cell's interaction list. All the cells in a level minus children of cells in the parent cell's interaction list is equivalent to the children of the parent's neighbors plus the children of the parent cell. This is at most $cn + c$. Excluding the cell's neighbors and the cell itself results in the expression

$$cn + c - n - 1 = (n + 1)(c - 1).$$

This expression is the maximum size of the interaction list. B_{M2L} is the maximum interaction list size times the maximum number of cells minus the root cell. The root cell is excluded because it is the only cell in level 0 and thus has an empty interaction list. Then,

$$B_{M2L} = (n+1)(c-1)\sum_{i=1}^{l-1} c^i = (n+1)(c-1)\frac{c^l - c}{c-1} = (n+1)(c^l - c). \tag{3.4}$$

The number of P2P operations is the number of near-field calculations performed. In the FMM, the near field of a source consists of all other sources in the source's cell and all sources in the source's cell's neighbors. Since P2P operations only involve leaf cells and there can be at most c^{l-1} leaf cells, then

$$B_{P2P} = [(n+1)s - 1]c^{l-1}. \tag{3.5}$$

Applying equations (3.2) to (3.5) into equation (3.1) gives

$$B = 2\frac{c^l - c}{c-1} + (n+1)(c^l - c) + [(n+1)s - 1]c^{l-1}. \tag{3.6}$$

3.1.2 Lowering the Upper Bound

In the FMM, the user can arbitrarily set l or s, *but not both*. The user cannot set both because they are related to the number of sources N, the distribution of sources, the partition used, and the number of child cells c each parent cell can have. Some FMM implementations, such as KIFMM [7] and exaFMM [8], typically let s be a free variable. Using this approach, we describe l in terms of N, c, s. If we assume sources are uniformly distributed in the problem region, then

$$l \approx \max\{\lceil \alpha[\log_c(N/s) + 1]\rceil, 1\} \tag{3.7}$$

where α is some postive constant.

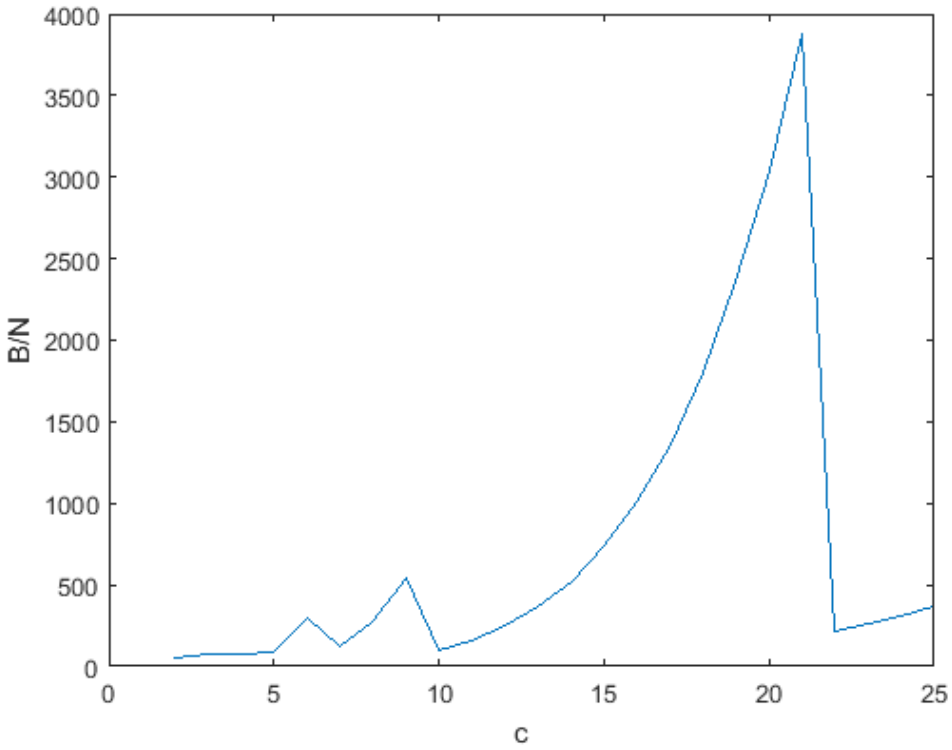


Figure 3: Normalized B for $N = 10,000$, $s = 1$, $\alpha = 1$, and $n = 8$.

We now consider options available for reducing B for a fixed N. The variable s is a free parameter set by the user and is independent of partition. This leaves c and n. To see how c affects B, let $N = 10,000$, $s = 1$, $\alpha = 1$, and $n = 8$ and plot B with respect to c. Setting $n = 8$ comes from the standard 2D FMM where every square cell can have up to eight neighbors. Figure 3 shows a plot of normalized B as a function of c. From equation (3.7), l is a monotonically decreasing function with respect to c. Since l only takes integer values, this results in B growing exponentially as c increases until l is reduced. While $c = 4 = 2^2$ and $c = 8 = 2^3$ are not necessarily local minima of B for all N, they are safe choices for 2D and 3D, respectively, and fixing c avoids the issue of designing partitions for different values of c. This leaves n, where equation (3.6) shows that B increases linearly with respect to n.

From the above analysis, we see that a viable means of reducing the number of partition-

dependent operations is to reduce the number of neighbors for each cell. However, determining the optimal partition for some n is not a trivial task. Fortunately, a look into group theory may produce possibly good solutions.

3.2 Relevant Concepts from Group Theory

While much of this thesis focuses on the fast multipole method, a look into in relevant group theory topics sets up more theoretical foundation for solutions in improving FMM performance.

3.2.1 Groups and Coxeter Systems

Group theory is the study of the algebraic structures called "groups." Fraleigh defines a group as follows [9].

Definition 3.1. *A **group** $\langle G, * \rangle$ is a set G with a binary operation $*$ on G such that the following axioms are satisfied:*

- *The binary operation $*$ is associative.*
- *There exists an identity element e in G such that $e * x = x * e = x, \forall x \in G$.*
- *There exists an element a' in G for each a in G with the property that $a' * a = a * a' = e$.*

Of particular interest are what are called Coexter systems, an abstract set of groups each generated by a set of generators introduced by H. S. M. Coxeter in 1934 [10, 11, 12].

Definition 3.2. *A **Coxeter group** is a group generated by a set of generators $\{r_1, r_2, ..., r_n\}$ and the relations $(r_i r_j)^{k_{ij}} = 1$ where $k_{ij} \in \mathbb{N}^+ \cup \{\infty\}$ and $k_{ij} = 1$ if $i = j$, $k_{ij} \geq 2$ if $i \neq j$, and $k_{ij} = \infty$ signifies r_i and r_j do not have the relationship $(r_i r_j)^{k_{ij}} = 1$* [10].

Definition 3.3. *A **Coxeter matrix** is an N x N symmetric matrix where k_{ij} of a Coxeter group is the (i, j) entry of the matrix* [12].

Definition 3.4. *If a set of generators S, along with some defining relations between the generators, produces a Coxeter group G, then the pair (G, S) is a **Coxeter system*** [12].

Coxeter systems are classified by the set of generators and the set of associated relations. This is because Coxeter groups alone do not uniquely identify the generators used [11]. Although there are several types of Coxeter systems, we focus only on type $\tilde{A}_n$ Coxeter systems, $n \geq 2$, which have the following properties [12]:

- There are $n + 1$ elements in the set of generators.
- $k_{ij} = 1$ if $i = j$.
- $k_{ij} = 2$ if $|i - j| > 1$ and $|i - j| \neq n$.
- $k_{ij} = 3$ if $|i - j| = 1$ or $|i - j| = n$.

Note that there are also type A_n Coxeter systems which are related but distinct from type $\tilde{A}_n$ Coxeter systems [10, 11, 12]. These A_n Coxeter systems are not germane to this thesis.

3.2.2 $\tilde{A}_n$ Coxeter System in Two and Three Dimensions

A geometric interpretation of Coxeter systems is that the set of generators represents a set of reflectors in some vector space [10, 12]. A reflector represented by r_i and a reflector represented by r_j have an "angle" of π/k_{ij} between them [10]. "Angle" is in quotation marks because Coxeter systems generally cannot be realized in Euclidean space [12]. Type $\tilde{A}_n$ Coxeter systems, however, can be realized in Euclidean n-space. Since we are only concerned with space in two and three dimensions, we only consider the $\tilde{A}_2$ and $\tilde{A}_3$ Coxeter systems.

Based on the properties of $\tilde{A}_n$ Coxeter groups listed earlier, the Coxeter matrix for the $\tilde{A}_2$ Coxeter group is

$$\begin{bmatrix} 1 & 3 & 3 \\ 3 & 1 & 3 \\ 3 & 3 & 1 \end{bmatrix}.$$

This Coxeter matrix defines the relations between three generators. The geometric interpretation of these relations is that in 2D Euclidean space, every pair of reflectors intersect

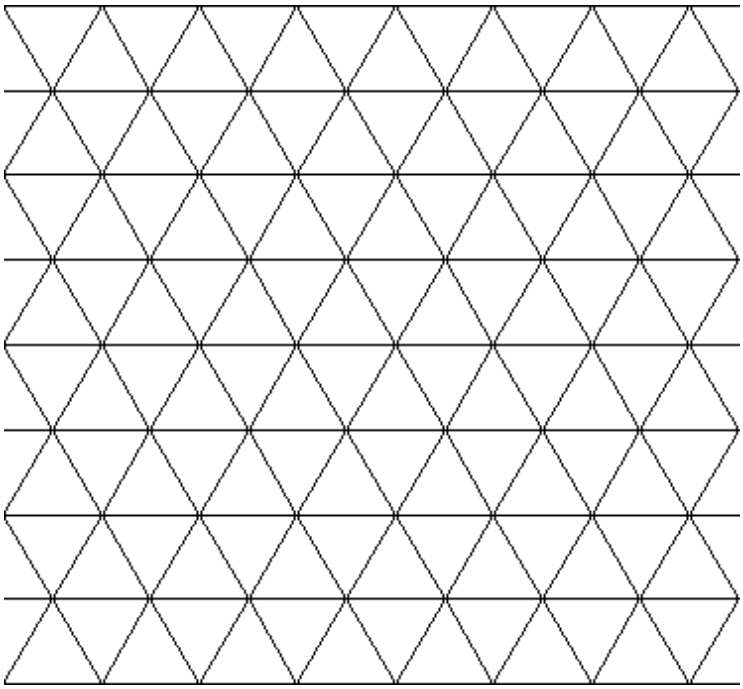


Figure 4: Triangular tesselation.

at an angle of $\frac{\pi}{3}$. That means the reflectors meet to form an equilateral triangle. The result is infinite mirror images that resemble the triangle tessellation shown in Figure 4 [10, 12].

For the $\tilde{A}_3$ Coxeter system, the Coxeter matrix is

$$\begin{bmatrix} 1 & 3 & 2 & 3 \\ 3 & 1 & 3 & 2 \\ 2 & 3 & 1 & 3 \\ 3 & 2 & 3 & 1 \end{bmatrix}.$$

This Coxeter matrix defines the relations between four generators. Here, the geometric interpretation is that in 3D Euclidean space, each reflector intersects two reflectors at an angle of $\frac{\pi}{3}$ and one reflector at an angle of $\frac{\pi}{2}$ [10, 13]. This results in the tetrahedral cell as shown in Figure 5. The cell is put in a reference cube to illustrate the geometric properties of this particular tetrahedron. The reflectors produce infinite mirror images that resemble the tetrahedral-octahedral honeycomb shown in Figure 6 [13].

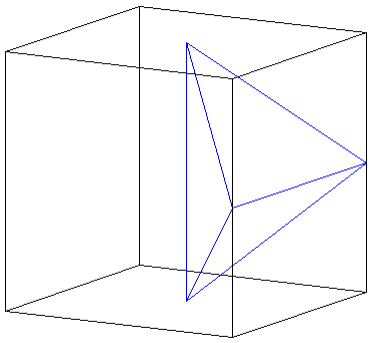


Figure 5: $\tilde{A}_3$ cell.

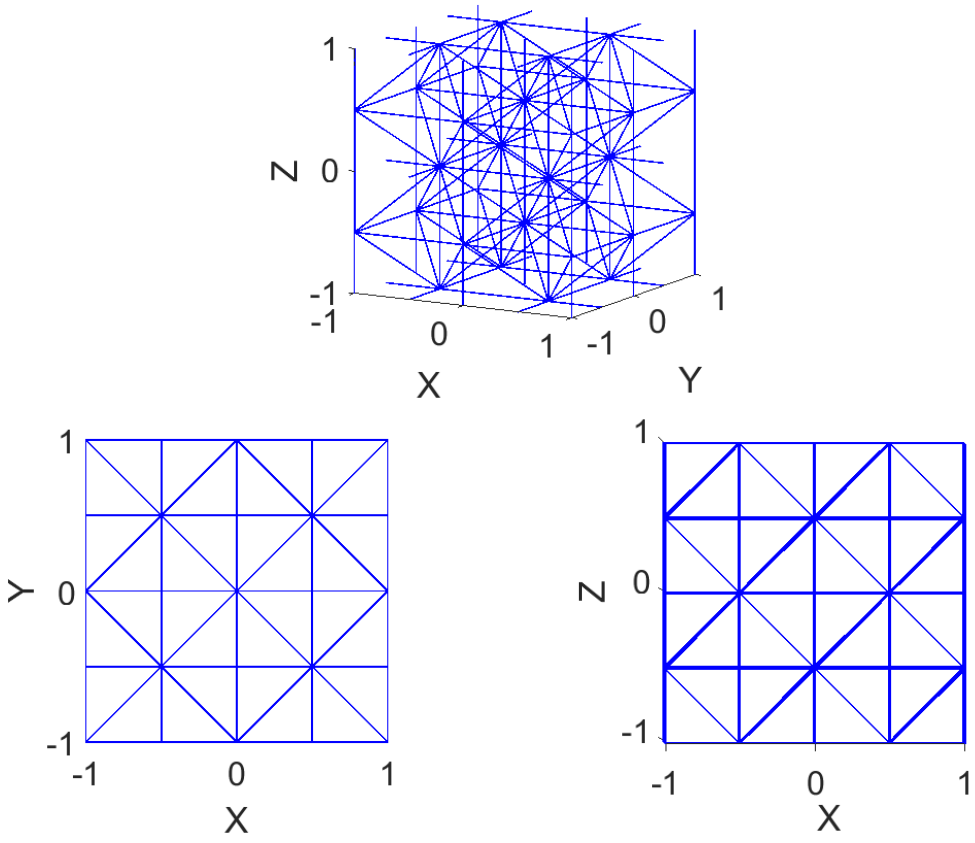


Figure 6: Three viewpoints of the tetrahedral-octahedral honeycomb.

Consider the intersection points in the geometric realizations of the $\tilde{A}_2$ and $\tilde{A}_3$ Coxeter systems. These sets of points are lattices with special properties in $\mathbb{R}^2$ and $\mathbb{R}^3$, respectively. These lattices will subsequently and respectively be called the $\tilde{A}_2$ lattice and the $\tilde{A}_3$ lattice.

Definition 3.5. *Given a set of basis vectors that span $\mathbb{R}^n$, a **lattice** is the set of all linear combinations with integer coefficients of the basis vectors* [14].

One important property of the $\tilde{A}_2$ and $\tilde{A}_3$ lattices is that these lattices are the optimal lattice solutions to the thinnest covering problem [14]. The thinnest covering problem seeks to find the most efficient way to cover $\mathbb{R}^n$ with equal overlapping spheres. A covering of spheres is considered more efficient than another covering of equally-sized spheres if the spheres in the former have less, or "thinner," overlap than the spheres in the latter [14].

3.3 Connecting FMM and Group Theory

We can think about the thinnest covering problem as finding the largest radius of space that can be covered given some number of identical spheres. This problem is similar to finding the best arrangement of cells in an FMM partition to maximize the smallest distance between the center of a cell and a point outside the area covered by the cell and its neighbors given the maximum number of neighbors a cell can have. Maximizing this distance is important because the error bound between the exact solution and the approximate solution gets smaller as the distance between the cells gets larger [1]. As the maximum number of neighbors for each cell decreases in a partition, the smaller the "near field" of a cell becomes. Thus, a tradeoff exists between decreasing computation time by decreasing the number of neighbors for each cell and increasing the error of the FMM. The $\tilde{A}_2$ and $\tilde{A}_3$ lattices offer good starting points for this tradeoff. Because the thinnest covering problem considers overlapping spheres and FMM partitions consider nonoverlapping cells, it is not conclusive the $\tilde{A}_2$ and $\tilde{A}_3$ lattices are the optimal solutions to the problem statement of this thesis. However, subsequent chapters will show that in fact FMM partitions whose cell centers form lattices similar to the $\tilde{A}_2$ and $\tilde{A}_3$ lattices provide significant decreases in computation time with little to no increase in error.

CHAPTER 4

A NEW PARTITION IN 2D FMM

This chapter presents an original partition solution for improving the FMM in two dimensions. The goal is to develop FMM algorithms that utilize partitions that meets design objectives for good partitions. Experiments will compare the performance of the Black-box FMM in 2D, or BBFMM2D [15], an open-source 2D FMM program by Fong and Darve, and the performance of BBFMM2D modified with the proposed 2D partition design. Performance will be judged based on computation time and relative error between the FMM approximation and the exact solution of the N-body problem.

4.1 Partition Design

Prior works discussed in Chapter 2 showed that previous research in FMM partitioning found that square cells generally work better than nonsquare cells for 2D problem regions [4, 5]. If the root cell is a square, then there is a straighforward way to divide the root cell recursively into smaller square cells. However, if the goal is to reduce the number of neighbors each cell has, then determining a recursive way to divide a square root cell without increasing complexity is nontrivial.

Chapter 3 showed that we can partition the problem region into at least one level of cells such that each cell gets fewer neighbors than in the standard quadtree method. The $\tilde{A}_2$ lattice offers a possible partition where each point in the lattice represents the center of a cell with only six neighbors compared to the eight neighbors in the standard quadtree. The problem with the $\tilde{A}_2$ lattice, however, is that if we put a square centered at each lattice point and use these squares to cover $\mathbb{R}^2$, as shown in Figure 7, then the squares would overlap and this layout would not be a partition. One approach would be to keep the $\tilde{A}_2$ lattice structure and use the Voronoi regions to get hexagonal cells, as shown in Figure 8.

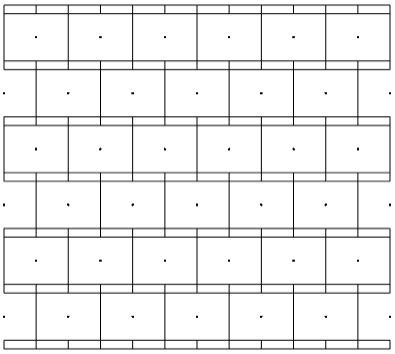


Figure 7: $\tilde{A}_2$ lattice with a square centered at each lattice point.

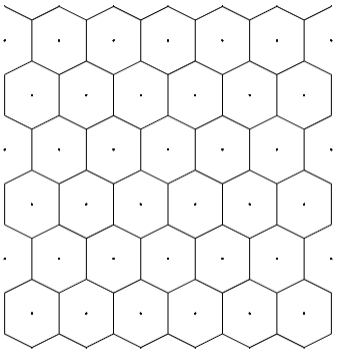


Figure 8: $\tilde{A}_2$ lattice with Voronoi regions.

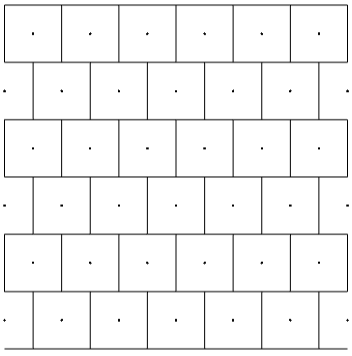


Figure 9: Nonoverlapping squares whose centers form a lattice different from the $\tilde{A}_2$ lattice.

The problem with this approach is that binning points[1] into hexagonal cells would involve checking diagonal boundaries, a process that is more complicated than checking horizontal and vertical boundaries. An alternative approach is to dilate the lattice points in one dimension such that square cells no longer overlap and thus, become a partition, as shown in Figure 9. Now we have a partition where every cell has only six neighbors and each cell is a square.

4.1.1 Minimizing the Number of Neighbors

We now prove that six is the absolute minimum of neighbors for square cells in 2D. We begin by formally defining some terms. All definitions in this chapter were independently developed for this thesis unless a citation is included.

Definition 4.1. *A **neighborhood** of a point $p \in \mathbb{R}^n$ is a set of points $S \subset \mathbb{R}^n$ where $\exists r > 0$ such that the set $\{x \in \mathbb{R}^n | \text{distance}(x, p) < r\} \subseteq S$* [16].

Definition 4.2. *The **interior** of a set of points $S \subset \mathbb{R}^n$ is the subset of points $\text{int}(S)$ where every point in $\text{int}(S)$ has a neighborhood that is a subset of S* [16].

Definition 4.3. *The **boundary** of a set of points $S \subset \mathbb{R}^n$ is the subset of points ∂S where every neighborhood of every point in ∂S has at least a point in S and at least a point not in S* [16].

Definition 4.4. *The **closure** of a set of points $S \subset \mathbb{R}^n$ is the union of S and the boundary of S* [16].

Definition 4.5. *A **simple rectilinear polygon** (SRP) is the closure of a set of points in $\mathbb{R}^2$ that has a single continuous boundary consisting of nonintersecting straight lines that connect together at right angles.*

Definition 4.6. *A **vertex** of an SRP is a corner boundary point.*

Definition 4.7. *An **edge** of an SRP is the set of boundary points between two vertices of the SRP.*

Definition 4.8. *Two SRPs A and B **overlap** if $\exists x \in \text{int}(A) : x \in \text{int}(B)$, or equivalently*

[1] Determining which points belong to which cells based on the cells' boundaries.

$\text{int}(A) \cap \text{int}(B) \neq \emptyset$.

Definition 4.9. *Two SRPs A and B are **neighbors** if* $\text{int}(A) \cap \text{int}(B) = \emptyset \wedge \partial A \cap \partial B \neq \emptyset$.

Definition 4.10. *Two SRPs A and B are **identical** if* $\exists x \in \mathbb{R}^2 : \forall a \in A, a + x \in B$.

Definition 4.11. *Given a set of N SRPs $S = \{S_1, S_2, ..., S_N\}$ whose union forms a set of points $T = S_1 \cup S_2 \cup ... \cup S_N$, an SRP $S_i \in S$ is **surrounded** if S_i is fully in the interior of T and S_i does not overlap with any other SRP in S.*

Definition 4.12. *A set of N SRPs $S = \{S_1, S_2, ..., S_N\}$ is a **partition** of a set of points T if no two SRPs in S overlap and the union of the SRPs in S forms T.*

Definition 4.13. *Suppose an SRP with area A has n vertices at points $(x_1, y_1), (x_2, y_2), ...,$ (x_n, y_n), and these points are in counterclockwise order along the boundary, then the **center** (x_c, y_c) of the SRP is given by* [17]

$$\begin{aligned} x_c &= \frac{1}{6A} \sum_{i=1}^{n} (x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \\ y_c &= \frac{1}{6A} \sum_{i=1}^{n} (y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \end{aligned} \tag{4.1}$$

where $x_{n+1} = x_1$ and $y_{n+1} = y_1$.

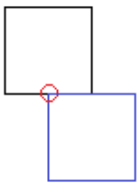
With the above definitions, we now prove that the least upper bound to the number of neighbors a square[2] cell can have in 2D is six.

Theorem 4.1. *A square is surrounded by six nonoverlapping squares identical to the surrounded square if each vertex of the surrounded square touches two nonoverlapping neighbors identical to the surrounded square.*

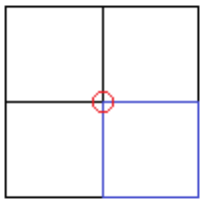
Proof. For a square to be surrounded, its vertices must be interior points in the union of the surrounded square and its neighbors. This means there exists a neighborhood around each vertex that contain points in this union of squares. Consider a circle of points that has as the center a vertex of a square and a radius that is infinitesimally small. Then, $\frac{1}{4}$ of the points in this circle comes from the originating square, and at most $\frac{1}{2}$ of the points in this

[2] A square is one type of an SRP.

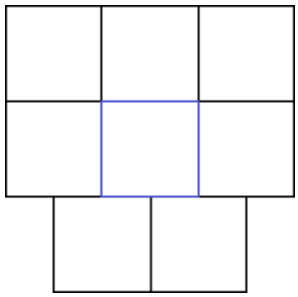
circle comes from a neighboring square. Thus, it is impossible for a vertex of a surrounded square to touch only one square neighbor. (See Figure 10a.) Each nonoverlapping square neighbor makes up at least $\frac{1}{4}$ of the points in this neighborhood. Thus, it is also impossible for a vertex to touch more than three nonoverlapping square neighbors. (See Figure 10b.) Suppose one vertex of a surrounded square touches three square neighbors. If two of the three remaining vertices of the surrounded square touch two square neighbors, then the last remaining vertex must touch three square neighbors, sharing one square neighbor with the first vertex. This results in a square surrounded by seven neighbors. (See Figure 10c.) To be surrounded by fewer neighbors, the only possibility left is for each vertex of a square to touch two square neighbors. This results in a square being surrounded by six square neighbors. (See Figure 10d.) □



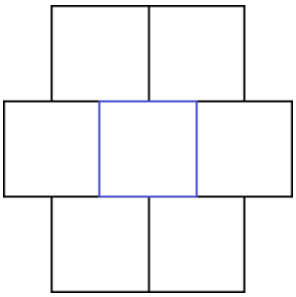
(a) One vertex (circled) of one square (blue) touching only one square neighbor (black).



(b) One vertex (circled) of one square (blue) touching three square neighbors (black).



(c) One square (blue) surrounded by seven neighbors.



(d) One square (blue) surrounded by six neighbors.

Figure 10: Illustrations for Theorem 4.1.

4.1.2 Maximizing Radial Covering

Given that the least upper bound in the number of nonoverlapping neighbors a square identical to its neighbors can have is six, the next step is to determine the best way to arrange these seven squares such that the smallest distance between the center of the surrounded square and a boundary point in the union of the seven squares is maximized. Maximizing this distance is important for minimizing the error upper bound between the far-field approximation solutions in the FMM and the exact solutions.

Theorem 4.2. *Consider the set of SRPs whose partitions each consists of a square with side length $2a$ surrounded by six squares identical to the surrounded square. For an SRP in the set, let $\mathcal{O}$ be the center of the surrounded square in the SRP's partition and r be the shortest distance between $\mathcal{O}$ and a boundary point of the SRP. The maximum possible value of r is $\sqrt{5}a$.*

Proof. From Theorem 4.1, any SRP in the set must have each vertex of the surrounded square touch two square neighbors. This can only be done using three rows (or three columns) of squares with the middle row having three squares. Fixing the position and orientation of the surrounded square, neither of the two neighboring squares in the middle row can move without breaking the partition. On the other hand, the top and bottom rows can move along the row up to where a square is aligned above or below the surrounded square, the point at which the surrounded square is no longer surrounded. An example of an SRP in the set is shown in Figure 11. From Figure 11, it is clear that there exists an SRP where $r = \sqrt{5}a$. From Figure 11, if the top or bottom row is moved, then $r < \sqrt{5}a$. Similar arguments apply to rotated versions of the SRPs discussed. □

4.1.3 An Efficient Partition with Square Cells

The next step is to show this arrangement of squares can be tessellated in $\mathbb{R}^2$, causing the centers of the squares to form a lattice.

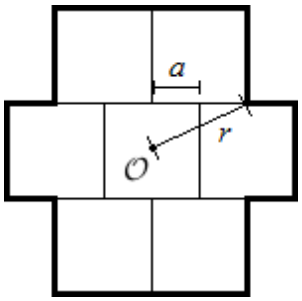


Figure 11: A partition of an SRP with the shortest distance between the center and a boundary point equalling $\sqrt{5}a$.

Definition 4.14. *An infinite set of SRPs is a* ***tessellation*** *if no two SRPs in the set overlap and the union of all SRPs in the set is* $\mathbb{R}^2$.

Definition 4.15. *A* $\mathbf{\Lambda_2}$ ***lattice*** *is a lattice in* $\mathbb{R}^2$ *generated by the basis vectors* v_1 *and* v_2 *where*

$$v_1 = \begin{bmatrix} 2a \\ 0 \end{bmatrix}, v_2 = \begin{bmatrix} a \\ 2a \end{bmatrix}, a \in \mathbb{R}. \tag{4.2}$$

Theorem 4.3. *Consider the set of SRPs whose partitions each consists of a square of side length* $2a$ *surrounded by six squares identical to the surrounded square. For an SRP in the set, let* $\mathcal{O}$ *be the center of the surrounded square in the SRP's partition and* r *be the shortest distance between* $\mathcal{O}$ *and a boundary point of the SRP. If an SRP in the set has the maximum possible value of* $r = \sqrt{5}a$*, then this SRP can be used as the basis of a tessellation. Further, the tessellation can be partitioned into a tessellation of squares whose centers form a* Λ_2 *lattice.*

Proof. From Theorem 4.2, the SRP in Figure 11 and all its rotated versions are the only SRPs in this set that have $r = \sqrt{5}a$. Figure 12 shows a tessellation of the SRP in Figure 11 exists and that this tessellation can be partitioned into a tessellation of squares whose centers form a Λ_2 lattice. Thus, Figure 12 proves the theorem. □

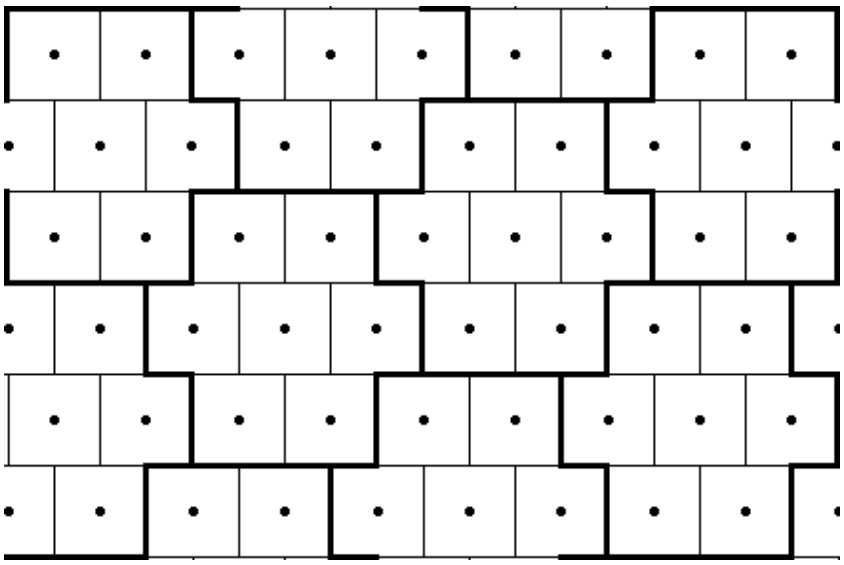


Figure 12: A tessellation of SRPs from Figure 11 that can be partitioned into squares whose centers form a Λ_2 lattice.

4.1.4 An Efficient Partition with SRPs in Higher Levels

We now have a way to partition a 2D problem region for one level. This partition is a set of squares that is a subset of the tessellation of squares whose centers form a Λ_2 lattice. This tessellation will be referred to as the set T_0. Determining a way to recursively divide these cells in a way that does not increase the number of neighbors for each cell while maintaining that each cell has some number of children is nontrivial. A process that is easier than determining finer levels of cells is determining coarser levels of cells. Unlike other FMM tree constructions where we start with a root cell and then generate levels of child cells until some criterion is met, we start with a partition of cells that cover the problem region and satisfy some criteria and generate levels of parent cells until a root cell is generated that covers the problem region. Since there is no clear benefit from changing the number of child cells each parent cell has, as explained in Chapter 3, parent cells in this partition design are generated such that they each have four child cells, just as parent cells in a quadtree structure each have four child cells. Since there is no way to take four square cells arranged in a Λ_2 lattice to create a square parent cell, it is important to consider

what is the best set of four cells should be used to create a parent cell. From findings in the past research of others in different FMM partitions, the parent cells should be as close to a square as possible [4, 6]. In other words, child cells in a parent cell should be as "compact" as possible. A new quantitative meaning of "compactness" is now presented.

Definition 4.16. *If an SRP is plotted in a Cartesian coordinate system such that each edge of the SRP is parallel to the x-axis or y-axis, then the **x-extent** of an SRP is the difference between the highest x-coordinate value of a point in the SRP and the lowest x-coordinate value of a point in the SRP.*

Definition 4.17. *If an SRP is plotted in a Cartesian coordinate system such that each edge of the SRP is parallel to the x-axis or y-axis, then the **y-extent** of an SRP is the difference between the highest y-coordinate value of a point in the SRP and the lowest y-coordinate value of a point in the SRP.*

Definition 4.18. *The **diagonal extent** of an SRP is the largest distance between two vertices of the SRP.*

Definition 4.19. *The **2D compactness** of an SRP with area A, x-extent X, y-extent Y, and diagonal extent D is given by the relation*

$$Q_2 \equiv \alpha_2 \beta_2 \tag{4.3}$$

where

$$\alpha_2 = \frac{A}{\max\{X, Y\}^2} \tag{4.4}$$

$$\beta_2 = \frac{\alpha_2}{F(D_0)} \tag{4.5}$$

$$D_0 = \frac{D}{\max\{X, Y\}} = \frac{1}{\cos\theta} \tag{4.6}$$

$$\begin{aligned} F(D_0) &= \frac{\pi D_0^2}{4} - 4\left(\frac{\pi D_0^2}{4}\frac{\theta}{\pi} - \frac{D_0^2}{4}\sin\theta\cos\theta\right) \\ &= \frac{\pi D_0^2}{4} - \left(D_0^2 \cos^{-1}\left(\frac{1}{D_0}\right) - D_0 \sin\left(\cos^{-1}\left(\frac{1}{D_0}\right)\right)\right). \end{aligned} \tag{4.7}$$

This definition of compactness Q_2 is designed to consider two factors. The first factor α_2 is the ratio between the area of the SRP and the area of the smallest square that can encompass the SRP. This factor serves as a bias against SRPs with a high value for max$\{X, Y\}$/min$\{X, Y\}$. The second factor β_2 is the ratio between α_2 and the maximum possible α_2 that a 2D shape with diagonal extent D inside a square with side length max$\{X, Y\}$ can have. This maximum possible α_2 is denoted by $F(D_0)$, where D_0 is the diagonal extent of an SRP scaled to fit inside a square with side length 1. $F(D_0)$ is then the maximum area that can be covered by a shape with diagonal extent D_0 inside a square of side length 1. This can be found by calculating the area of the intersection between a square with side length 1 and a circle with diameter D_0 concentric with the square. This factor serves as a bias against SRPs that are longer in one diagonal direction over another diagonal direction. Thus, $Q_2 \leq 1$, and $Q_2 = 1$ iff the SRP is a square.

The goal now is to maximize Q_2. If each parent cell has four child cells, then the area of a cell is $4^n s^2$, where s is the side length of a square at the finest partition, and n is the level number with level 0 being the *finest* level. This labeling of the different levels is contrary to the standard FMM, where level 0 is the root level. Consider the partition of squares in Figure 12. There are multiple ways to form a level 1 parent cell, or l_1 *cell*, with four contiguous child cells. An extreme example is one row of four squares with $Q_2 = 0.0758$. An exhaustive search gives two possible solutions with $Q_2 = 0.417$ as shown in Figure 13.

Since the two solutions are simply reflections of each other, either one can be chosen. The solution on the right will be used for subsequent discussion and will be referred to as

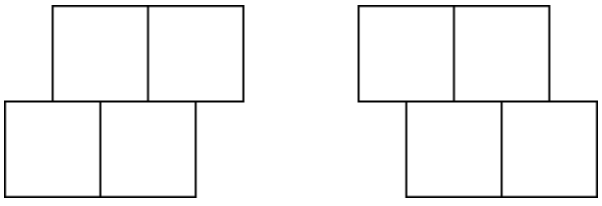


Figure 13: Optimal l_1 cells.

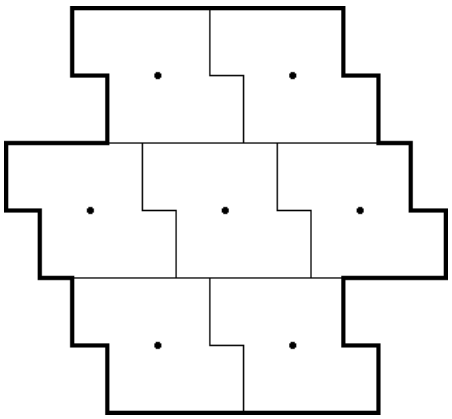


Figure 14: SRP with a partition of l_1 cells and maximum r.

the l_1 cell. Consider the set of SRPs whose partitions consist of an l_1 cell surrounded by six l_1 cells identical to the surrounded cell, and these partitions can be further partitioned to a subset of T_0. For an SRP in the set, let $\mathcal{O}$ be the center of the surrounded cell and r be the shortest distance between $\mathcal{O}$ and a boundary point of the SRP. The next step is to maximize r. Because of the restriction that the SRP must be able to be partitioned to a subset of T_0, the set is finite. It turns out that there are seven SRPs in the set, and the SRP that maximzes r is the one shown in Figure 14. To see the seven possibilities, let x be the center of an l_1 cell and v_k be a vector that describes the difference between the center of the kth neighbor of the cell and x. If the side length of the l_0 cells used as the basis for the l_1 cells is 1, then the seven distinct possible sets of $\{v_k\}$ are those as described in Table 1. An interesting observation is that the centers of the l_1 cells in this partition form another Λ_2 lattice.

Table 1: Seven ways to surround an l_1 cell with six identical cells formed by l_0 cells in T_0.

	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7
v_1	$(-2.5, 1)$	$(-2.5, 1)$	$(-2, 0)$	$(-2, 0)$	$(-2, 0)$	$(-2, 0)$	$(-2, 0)$
v_2	$(-1.5, -1)$	$(-1.5, -1)$	$(-1, -2)$	$(-1, -2)$	$(-1, -2)$	$(0, -2)$	$(0, -2)$
v_3	$(1, -2)$	$(1, -2)$	$(1, -2)$	$(1, -2)$	$(1, -2)$	$(2, -2)$	$(2, -2)$
v_4	$(2.5, -1)$	$(2, 0)$	$(2.5, -1)$	$(2, 0)$	$(2, 0)$	$(2, 0)$	$(2, 0)$
v_5	$(1.5, 1)$	$(1, 2)$	$(1.5, 1)$	$(1, 2)$	$(0, 2)$	$(0, 2)$	$(1, 2)$
v_6	$(-1, 2)$	$(-1, 2)$	$(-1, 2)$	$(-1, 2)$	$(-2, 2)$	$(-2, 2)$	$(-1, 2)$

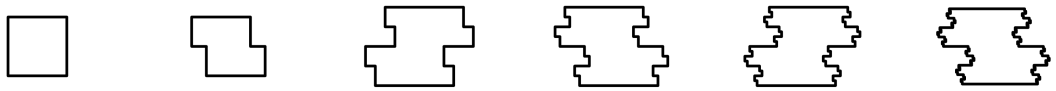


Figure 15: 2D l_n cells for $n \leq 5$.

There is now an iterative process for determining different levels of partitions in FMM. The first step is to maximize Q_2 of an l_n cell while maintaining a partition of the problem region in l_{n-1} cells. In the base case, Q_2 is maximized when the cell is a square. The second step is to determine the arrangement of l_n cells that maximizes the shortest distance r between the center of an l_n cell with a boundary point of the union of the cell and its neighbors. Theorem 4.3 shows that in level 0, the cells are arranged such that the centers form a Λ_2 lattice. In fact, r is maximized when the l_n cells are arranged such that the centers form an Λ_2 lattice. This arrangement sets up the basis for determining the l_{n+1} cell. In the final iteration, when the maximum level is reached, only maximizing Q_2 is necessary because the resulting cell would be the root cell and the root cell has no neighbors. Figure 15 shows the first six l_n cells.

4.2 Implementation

In the standard 2D FMM, the problem region is placed in an encompassing square root cell to be partitioned into smaller sizes of square cells in a quadtree structure. Generating these squares and binning source data is easy to implement. However, the new 2D partition proposed requires a separate piece of code to determine the boundaries of FMM cells at different levels. These boundaries only need to be determined once, and then these cells can be stored as a look-up table of vertices in counterclockwise order along the boundary. A pattern emerges when one manually uses the iterative process described in the previous section to determine an l_n cell. The pattern is that for $n \geq 1$, l_n is made up of four l_{n-1} cells

grouped together such that the centers of the l_{n-1} cells alternate between the relationship

$$\begin{aligned} a &= (x, y), & b &= (x + 2^{n-1}s, y), \\ c &= (x - 2^{n-2}s, y + 2^{n-1}s), & d &= (x + 2^{n-2}s, y + 2^{n-1}s) \end{aligned} \tag{4.8}$$

and

$$\begin{aligned} a' &= (x', y'), & b' &= (x' - 2^{n-1}s, y'), \\ c' &= (x' - 2^{n-2}s, y' + 2^{n-1}s), & d' &= (x' + 2^{n-2}s, y' + 2^{n-1}s) \end{aligned} \tag{4.9}$$

where x, y, x', and y' are arbitrary values, and s is the side length of the l_0 cell. Assuming this pattern holds, l_n cells are generated using Algorithm 1, GENERATECELLS. GENERATECELLS is created as the first step to constructing FMM trees with up to $m + 1$ levels under the proposed design criteria.

The next step is to bin points into cells. With a square or rectangular parent cell, binning points into child cells only require checking if the x- and y-coordinates of the points are greater or less than some x and y values that represent the vertical and horizontal lines that divide the parent cell into child cells. With the cells generated by GENERATECELLS, two complications need to be addressed. First, the positions of the top two child cells relative to the positions of the bottom two child cells alternate as the level number increases. Thus, logical operations involving the level number are needed. Here, checking if the level number is even or odd is sufficient. Second, the boundary between two sidy-by-side child cells is not generally a vertical line. Thus, logical operations involving the x-coordinate of a point depend on the y-coordinate of the point and the vertices of the child cells. From Figure 15, it appears that the boundary of an l_n cell consists of 2^n vertical lines of length equal to the side length of an l_0 cell left of the l_n cell's center and another 2^n vertical lines right of the center. Since a cell's vertices to the right of its center are equivalent to the cell's right-hand neighbor's vertices to the left of the neighbor's center, only the vertices left or right of the cell's center need to be known. This means assigning a point to an l_n cell involves using the point's y-coordinate to determine which of 2^n vertical lines to compare

Algorithm 1 – GENERATECELLS An algorithm for generating l_n cells up to maximum level m each as an array of vertices in counterclockwise order along the cell's boundary.

Input: maximum level m
Output: list of arrays of vertices with the vertices in counterclockwise order

$p_0 := \{(-0.5, -0.5), (0.5, -0.5), (0.5, 0.5), (-0.5, 0.5)\}$
$P := \{p_0\}$
if $m = 0$ **then**
    return P
end if
children $:= \emptyset$
for $i := 0, 1, ..., m - 1$ **do**
    if i is even **then**
        children $:= \{(0, 0), (-2^i, 0), (-2^{i-1}, -2^i), (2^{i-1}, -2^i)\}$
    else
        children $:= \{(0, 0), (-2^i, 0), (2^{i-1}, 2^i), (-2^{i-1}, 2^i)\}$
    end if
    $p_a := p_i + \text{children}[1]$ *// p_a is every element of p_i translated by coordinate children[1]*
    $p_b := p_i + \text{children}[2]$
    $p_c := p_i + \text{children}[3]$
    $p_{i+1} := (p_i \cup p_a \cup p_b \cup p_c) \setminus ((p_i \cap p_a) \cup (p_i \cap p_b) \cup (p_i \cap p_c) \cup (p_a \cap p_b) \cup (p_a \cap p_c) \cup (p_b \cap p_c))$
    // vertices shared between child cells are not vertices in the union of the child cells
    $\theta := \pi$
    for $j := 0, 1, ...,$ size of $p_{i+1} - 1$ **do** *// find a starting point*
        if $\tan^{-1}(p_{i+1}[j]) < \theta$ **then**
            $\theta := \tan^{-1}(p_{i+1}[j])$
            $\text{swap}(p_{i+1}[0], p_{i+1}[j])$
        end if
    end for
    for $j := 0, 1, ...,$ size of $p_{i+1} - 2$ **do** *// arrange points in counterclockwise order*
        $\theta := \pi$
        for $k := j + 1, j + 2, ...,$ size of $p_{i+1} - 1$ **do**
            if $p_{i+1}[j].x = p_{i+1}[k].x$ **or** $p_{i+1}[j].y = p_{i+1}[k].y$ **then**
                if $\tan^{-1}(p_{i+1}[k]) < \theta$ **then**
                    $\theta := \tan^{-1}(p_{i+1}[k])$
                    $\text{swap}(p_{i+1}[j + 1], p_{i+1}[k])$
                end if
            end if
        end for
    end for
    $(x_c, y_c) := \text{center}(p_{i+1})$ *// use equation 4.1*
    $p_{i+1} := p_{i+1} - (x_c, y_c)$ *// make cell centered at (0,0)*
    $P := P \cup p_{i+1}$
end for
return P

with the point's x-coordinate. A new algorithm that assigns points from a parent cell to child cells in the proposed partition is presented in Algorithm 2, BINTO2DCELLS. Figure 16 gives an illustrative example of what BINTO2DCELLS does when assigning a point from a parent l_3 cell to a child l_2 cell.

In the standard FMM, it is possible to allow levels to be continuously created until all cells in the most recently created level each has less than some user prescribed number of points s. This is because the child cells are simply scaled versions of the parent cell. This is not the case in this modified FMM. Thus, any implementation of this FMM requires approximating the number of levels needed before points are binned. Generally, $\lceil \log_4(N/s)+1 \rceil$ levels are enough for uniformly distributed points, and $\lceil \log_4(N/s)+2 \rceil$ or so levels are enough for nonuniformly distributed points. While leaf cells not in the finest level of cells are guaranteed to have less than s points, no such guarantee can be made for cells in the finest level.

The last step is defining what a cell's neighbors are and what cells are members of a cell's interaction list. Some FMM implementations use explicit lists to determine which child cells of a cell's parent and those of the cell's parent's neighbors are neighbors of the cell, while other implementations search for cells whose centers are within some distance away from a cell's center to get a list of a cell's neighbors [8, 15, 18]. Regardless, there are no substantial algorithmic changes needed for this enhanced FMM as either the explicit lists just need the entries changed or the nature of the cells' position is enough to determine each cell's neighbors. Because of the definition of the interaction list, interaction lists change automatically as neighbors change.

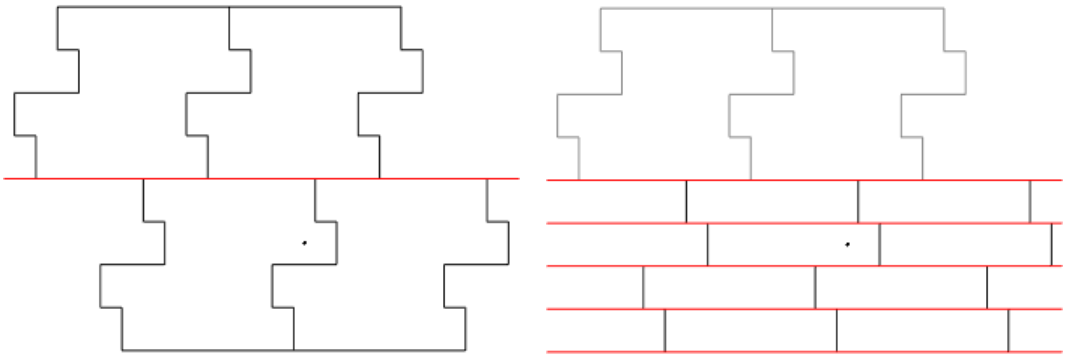
4.3 Experimental Setup

An open-source 2D FMM code created by Fong and Darve called the Black-box Fast Multipole Method in 2D, or BBFMM2D [15], is used as a reference point for 2D FMM performance. This FMM implementation is compared with a modified version that uses the proposed 2D partition. BBFMM2D uses Chebyshev interpolation and Chebyshev ap-

Algorithm 2 – BINTO2DCELLS An algorithm for assigning points in 2D from a parent l_n cell to child l_{n-1} cells

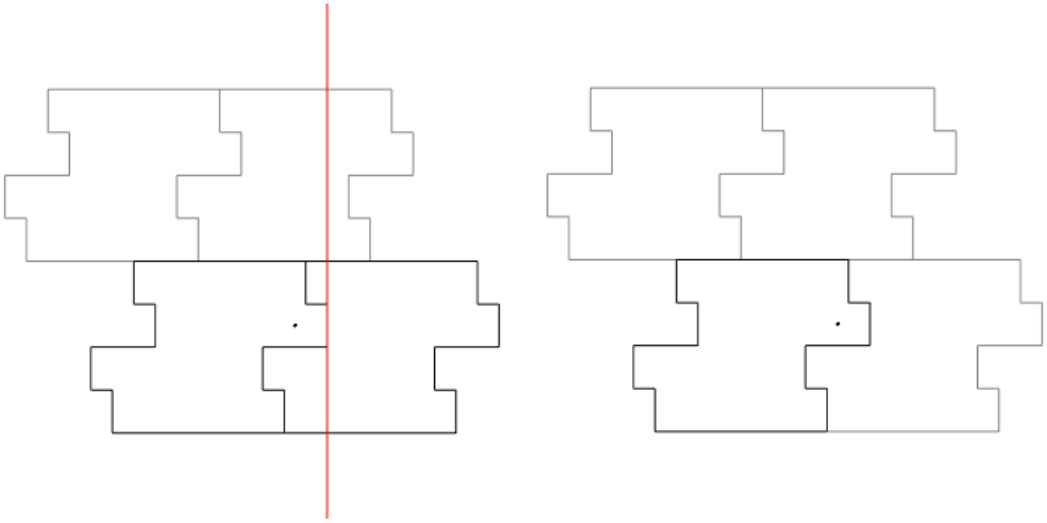
Input: level number n,
array P of vertices of an l_{n-1} cell centered at $(0,0)$ in counterclockwise order,
array X of N points in the l_n cell,
center (x_c, y_c) of the l_n cell,
y-extent Y_n of the l_n cell,
y-extent Y_0 of the l_0 cell
Output: an array of points for each child cell

$P^+ := \{(x,y) \in P \mid x > 0\}$ *// vertices to be used in boundary checks*
$(x_1, y_1) := (x_c, y_c) + ((-1.5 + n \mod 2)Y_n/4, -Y_n/4)$ *// centers of the child cells*
$(x_2, y_2) := (x_c, y_c) + ((0.5 + n \mod 2)Y_n/4, -Y_n/4)$
$(x_3, y_3) := (x_c, y_c) + ((-0.5 - n \mod 2)Y_n/4, Y_n/4)$
$(x_4, y_4) := (x_c, y_c) + ((1.5 - n \mod 2)Y_n/4, Y_n/4)$
$X_1 := \emptyset, X_2 := \emptyset, X_3 := \emptyset, X_4 := \emptyset$
for $i := 0, 1, ..., N-1$ **do**
    $(x,y) := X[i]$
    if $y < y_c$ **then**
        $\text{index} := 2\lfloor y - y_1 + Y_n/4 \rfloor / Y_0$
        $(x', y') := P^+[\text{index}] + (x_1, y_1)$
        if $x < x'$ **then**
            $X_1 := X_1 \cup (x,y)$
        else
            $X_2 := X_2 \cup (x,y)$
        end if
    else
        $(x', y') := (0,0)$
        if $y - y_3 = Y_n/4$ **then**
            $(x', y') := P^+[Y_n/Y_0 - 1] + (x_3, y_3)$
        else
            $\text{index} := 2\lfloor y - y_1 + Y_n/4 \rfloor / Y_0$
            $(x', y') := P^+[\text{index}] + (x_3, y_3)$
        end if
        if $x < x'$ **then**
            $X_3 := X_3 \cup (x,y)$
        else
            $X_4 := X_4 \cup (x,y)$
        end if
    end if
end for
return $\{X_1, X_2, X_3, X_4\}$



(a) Check if point is above or below the red horizontal line.

(b) Determine in which two red horizontal lines the point is between.



(c) Check if point is left or right of the red vertical line.

(d) The child cell the point is in has been determined.

Figure 16: Assigning a point from a parent l_3 cell to a child l_2 cell.

proximation theory [15]. This method is based on using the roots of a degree-k Chebyshev polynomial over the interval $[-1, 1]$ [19]. The roots of a degree-k Chebyshev polynomial scaled to an arbitrary interval $[a, b]$ is [19]

$$x_i = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{2i-1}{2k}\pi\right), i = 1, 2, ..., k. \tag{4.10}$$

Instead of each cell having just one multipole, each cell has k^2 multipoles with this approach. Figure 17 gives an example for $k = 5$. In our enhanced FMM solution, this means each parent l_n cell with $N > k^2$ points, center (x_c, y_c), and y-extent Y_n is represented by k^2 multipoles in the interval $[x_c - Y_n/2, x_c + Y_n/2] \times [y_c - Y_n/2, y_c + Y_n/2]$. This method performs better when points cover cells versus points covering only some regions in the cells [15]. While this is good for a square problem region in the standard FMM, this is detrimental for a square problem region in our modified FMM because none of the l_n cells for $n \geq 1$ is square. As a compromise, a rectangular problem region is used. Figure 18 gives an example of a rectangular problem region with uniformly distributed points in a square root cell versus an l_3 root cell. The problem region is not centered in the square root cell because two fully covered child cells perform better with this approach than four partially covered child cells.

The tests use a rectangular problem region that is twice as long vertically as it is long horizontally and the kernel function $\mathbf{r}^{-2}$, where $\mathbf{r}$ is the Euclidean distance between two points. Because BBFMM2D is kernel-independent with different kernels behaving similarly in computation time and similarly in relative error except in scale [15], only one kernel is tested. One set of tests uses uniformly distributed points. Another set uses Gaussian distributed points, where the mean is at the center of the problem region, the x values have standard deviation 0.194 times the horizontal length of the problem region, and the y values have standard deviation 0.194 times the vertical length of the problem region. These standard deviation values were chosen because there would be a 1% chance an x value is

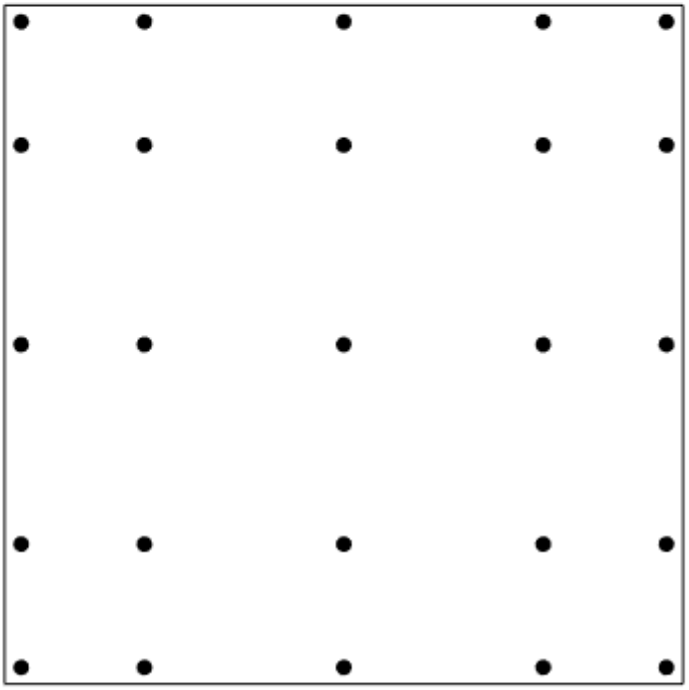


Figure 17: A cell with 25 multipoles using Chebyshev interpolation of degree 5.

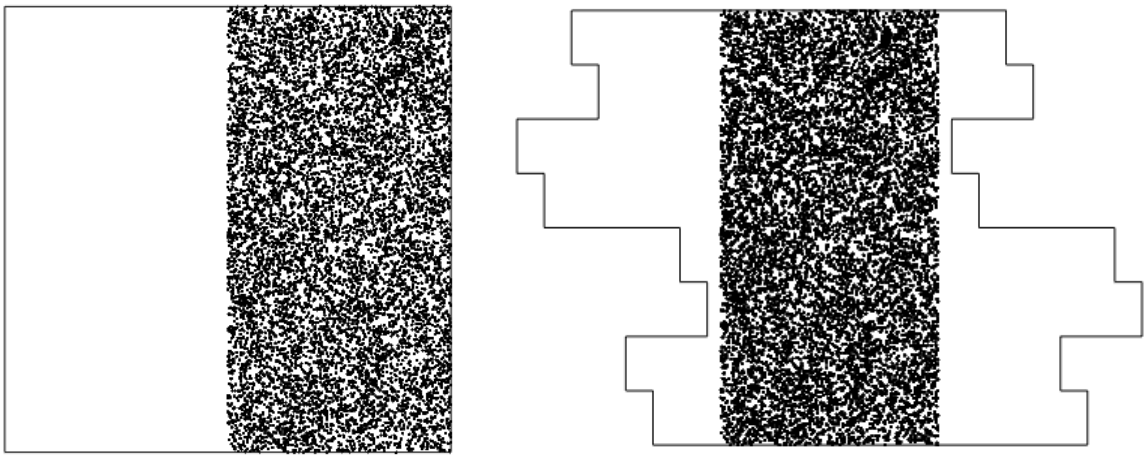


Figure 18: A rectangular problem region with uniformly distributed points in a square root cell (left) and an l_3 root cell (right).

generated outside the problem region and a 1% chance a y value is generated outside the problem region. For measuring computation time, the number of points in the problem region ranges from 10^4 to 10^6. This range of values is consistent with the range used in prior works [8, 15, 18]. For measuring error, the number of points ranges from 10^4 to 10^5. The number of points for error evaluation does not go beyond 10^5 because error calculation has complexity $O(N^2)$. For each value used for N, an average value from 10 Monte Carlo runs is used to measure computation time and evaluate relative error. The original version of BBFMM2D uses $k = 7$. The modified version uses $k = 8$ where error is lower compared to $k = 7$ in the modified version in exchange for more computation time.

4.4 Results

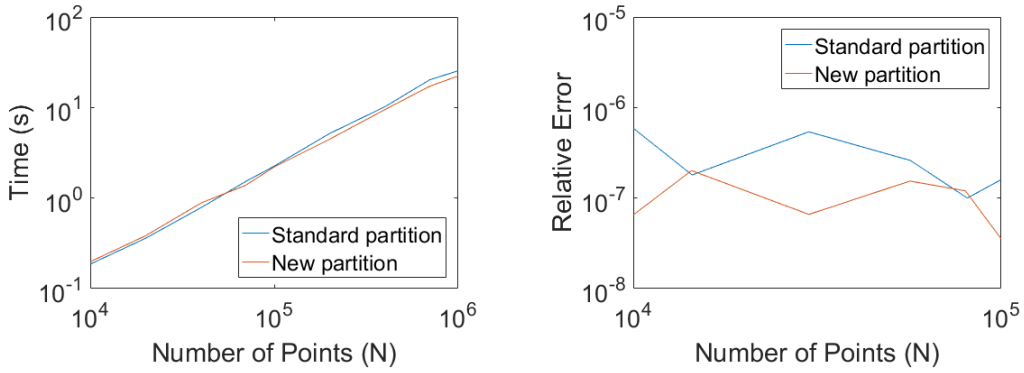
Figure 19a shows a plot comparing computation time between BBFMM2D with the standard partition and BBFMM2D with the new 2D partition for uniformly distributed points. For approximately $N < 50,000$, the computation time with the new partition is higher because the time savings from reducing the number of near-field and far-field calculations did not offset the additional time costs from a more complicated tree construction algorithm and from using $k = 8$ instead of $k = 7$ to maintain similar levels of error. For approximately $N > 50,000$, the savings from fewer near-field and far-field calcuations offset the costs from tree construction, resulting in as much as a 13% reduction in computation time. This is despite more work done per calculation with the new partition compared to the work done per calculation with the old partition.

Figure 19b shows a plot comparing relative error between BBFMM2D with the standard partition and BBFMM2D with the new 2D partition for uniformly distributed points. There is a large variation for the relative error, and there does not seem to be a pattern with respect to N. In general, the new partition results in a higher error than the standard partition does. This is expected because the lower covering of space by a cell and its neighbors reduces what is considered the near field, resulting in greater error between approximate values and exact values of far-field calculations. However, the range of error values in the new

partition overlaps with the range of error values in the standard partition.

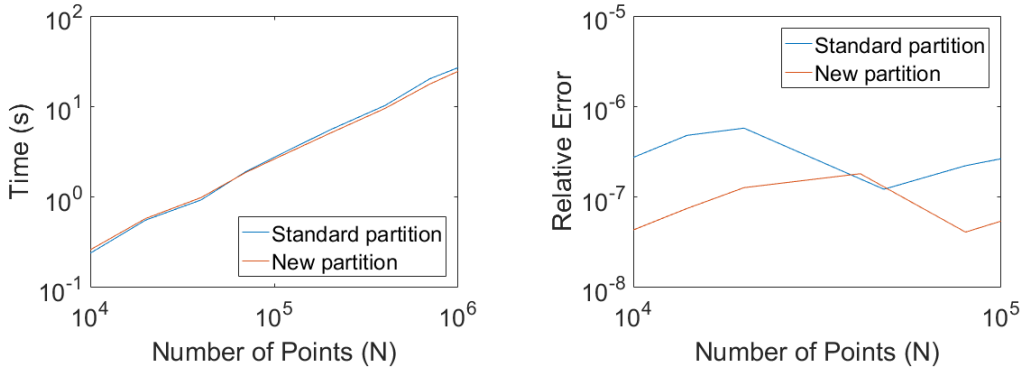
Figure 20 shows plots comparing computation time and relative error for Gaussian distributed points. These figures show that Gaussian distributed points behave similarly to uniformly distributed points. With Gaussian distributed points, computation time was reduced by as much as 9% for $N > 50,000$, a little less than the 13% for uniformly distributed points.

In 2D, our modified FMM did not produce much improvement in terms of lowering computation time while minimizing error. However, we will see that the number of neighbors each cell has can be more significantly reduced in 3D than in 2D, resulting in more computation time reduction.



(a) Computation time (b) Relative error

Figure 19: Standard partition vs. new 2D partition for uniformly distributed points.



(a) Computation time (b) Relative error

Figure 20: Standard partition vs. new 2D partition for Gaussian distributed points.

CHAPTER 5

A NEW PARTITION IN 3D FMM

This chapter presents an original partition solution for improving the FMM in three dimensions. The partition design in 3D largely will draw from the theory and methodology used in the 2D case. Experiments will compare the performance of exaFMM [8], an open-source 3D FMM program by Yokota, and the performance of exaFMM modified with the proposed 3D design. Performance will be judged based on computation time and relative error between the FMM approximation and the exact solution of the N-body problem.

5.1 Partition Design

Just as prior works discussed in Chapter 2 showed square cells generally work better than nonsquare cells for 2D problem regions, they also showed cubic cells generally work better than noncubic cells for 3D problem regions [4]. However, there is not a trivial solution to recursively divide a cubic root cell that guarantees the maximum number of neighbors each cell can have is lower than that in the standard octree without increasing complexity. Thus, we work our way up from the finest level of cells just as was done in 2D.

A good starting point is the $\tilde{A}_3$ lattice, because it is the optimal lattice solution for covering in $\mathbb{R}^3$ [14]. If a cube is centered at each point in the $\tilde{A}_3$ lattice, then the cubes have to overlap to cover $\mathbb{R}^3$. If Voronoi regions are used, then we get cells with diagonal faces [14]. Diagonal faces add binning complications in implementation that rectilinear cells would not. Another option is to dilate the lattice points in one dimension such that cubic cells no longer overlap and become a partition, as was done in the 2D case. This option results in an arrangement of cubes as shown in Figure 21. Now we have a partition where every cell has 16 neighbors, fewer than the maximum of 26 neighbors each cell has in an octree structure. The centers of these cubes in this arrangement represent a lattice slightly different from the $\tilde{A}_3$ lattice that will from now on be referred to as the Λ_3 lattice.

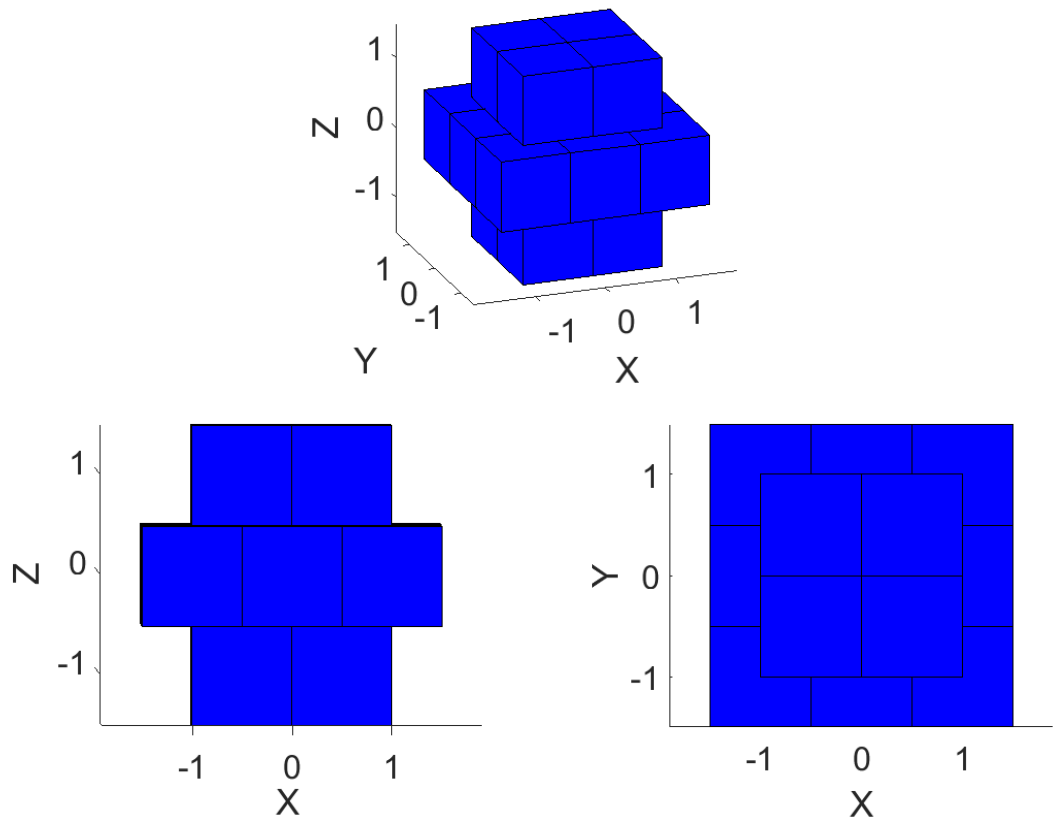


Figure 21: Three viewpoints of a cube with 16 neighboring cubes.

5.1.1 An Efficient Partition with Cubic Cells

We now formally define some terms. These include terms that were previously defined for the 2D case and are now extended to the 3D case. All definitions in this chapter were independently developed for this thesis unless a citation is included.

Definition 5.1. *A* $\mathbf{\Lambda_3}$ ***lattice*** *is a lattice in* $\mathbb{R}^3$ *generated by the basis vectors* v_1, v_2, *and* v_3 *where*

$$v_1 = \begin{bmatrix} 2a \\ 0 \\ 0 \end{bmatrix}, v_2 = \begin{bmatrix} 0 \\ 2a \\ 0 \end{bmatrix}, v_3 = \begin{bmatrix} a \\ a \\ 2a \end{bmatrix}, a \in \mathbb{R}. \tag{5.1}$$

Definition 5.2. *A* ***simple orthogonal polyhedron*** *(SOP) is the closure of a set of points in*

$\mathbb{R}^3$ *that has a single continuous boundary consisting of nonintersecting flat surfaces that connect together at right angles.*

Definition 5.3. *A **vertex** of an SOP is a corner boundary point.*

Definition 5.4. *An **edge** of an SOP is the set of boundary points between two vertices of the SOP.*

Definition 5.5. *A **face** of an SOP is the set of boundary points bounded by coplanar edges of the SOP.*

Definition 5.6. *Two SOPs A and B **overlap** if $\exists x \in \text{int}(A) : x \in \text{int}(B)$, or equivalently $\text{int}(A) \cap \text{int}(B) \neq \emptyset$.*

Definition 5.7. *Two SOPs A and B are **neighbors** if $\text{int}(A) \cap \text{int}(B) = \emptyset \wedge \partial A \cap \partial B \neq \emptyset$.*

Definition 5.8. *Two SOPs A and B are **identical** if $\exists x \in \mathbb{R}^3 : \forall a \in A, a + x \in B$.*

Definition 5.9. *Given a set of N SOPs $S = \{S_1, S_2, ..., S_N\}$ whose union forms a set of points $T = S_1 \cup S_2 \cup ... \cup S_N$, an SOP $S_i \in S$ is **surrounded** if S_i is fully in the interior of T and S_i does not overlap with any other SOP in S.*

Definition 5.10. *A set of N SOPs $S = \{S_1, S_2, ..., S_N\}$ is a **partition** of a set of points T if no two SOPs in S overlap and the union of the SOPs in S forms T.*

In 2D, the center of any simple rectilinear polygon is given by equation 4.1. Although an analogous version exists in 3D [20], because of the scope of this thesis, a narrower, simpler definition applying to simple orthogonal polyhedrons that can be partitioned into identical cubes[1] is provided.

Definition 5.11. *The **center** (x_c, y_c, z_c) of a cube with eight vertices (x_1, y_1, z_1), (x_2, y_2, z_2), ..., (x_8, y_8, z_8) is given by*

$$(x_c, y_c, z_c) = \frac{1}{8} \sum_{i=1}^{8} (x_i, y_i, z_i). \tag{5.2}$$

*The **center** (x_c, y_c, z_c) of an SOP that can be partitioned into N identical cubes with centers*

[1] A cube is one type of an SOP.

(x_1, y_1, z_1), (x_2, y_2, z_2), ..., (x_N, y_N, z_N) *is given by* [21]

$$(x_c, y_c, z_c) = \frac{1}{N} \sum_{i=1}^{N} (x_i, y_i, z_i). \tag{5.3}$$

One key difference arises between the Λ_2 lattice and the Λ_3 lattice. The Λ_2 lattice of squares has each cell with the minimum number of neighbors a surrounded cell could have. This is not the case with the Λ_3 lattice of cubes. For example, Figure 22 shows an arrangement of cubes where the surrounded cube has 14 neighbors. From equation 3.6, the arrangement of cells in Figure 22 would theoretically allow the FMM to run faster than with the Λ_3 lattice. However, another desirable attribute that has to be kept in mind is the shortest distance r between the center of a cube and a boundary point of the union of the cube and its neighbors. This value is important because the lower r is, the more the FMM approximation error can increase, as explained in Section 3.3. With the Λ_3 lattice, $r = \sqrt{5}s/2$, where s is the side length of a cube. With the arrangement in Figure 22, $r = \sqrt{13}s/4$. Reducing the number of neighbors from 16 to 14 may not be worth reducing r by about 20%. Thus, we choose to proceed in our 3D algorithm development with the Λ_3 lattice.

5.1.2 An Efficient Partition with SOPs in Higher Levels

We now consider how to build a parent cell from eight child cells. We consider eight because this is the same number of child cells each parent cell has in an octree structure. There is no clear benefit in using a different number of child cells each parent cell has, as explained in Section 3.1. However, there is no way to take eight cubic cells arranged in a Λ_3 lattice to create a cubic parent cell. From findings in the past research of others into different FMM partitions, the parent cells should be as close to a cube, or "compact", as possible [4, 6]. A quantitative meaning of "compactness" for 3D analogous to the 2D version is now presented.

Definition 5.12. *If an SOP is plotted in a Cartesian coordinate system such that each edge*

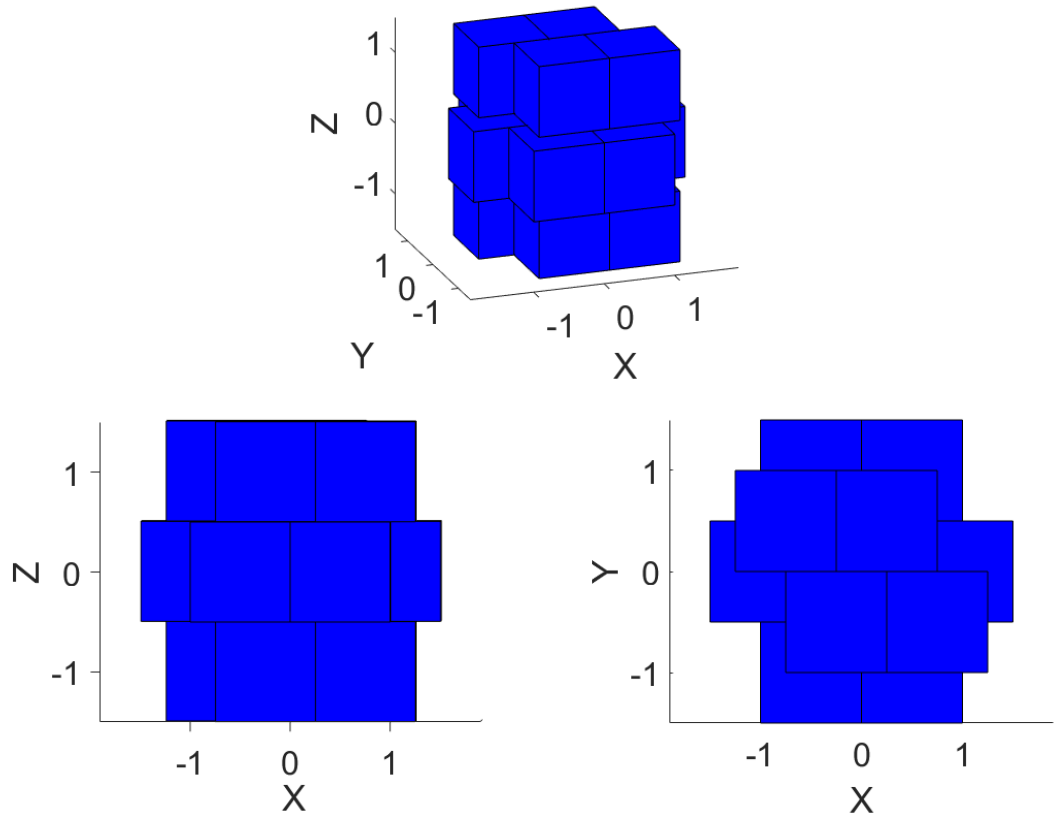


Figure 22: Three viewpoints of a cube with 14 neighboring cubes.

of the SOP is parallel to the x-axis, y-axis, or z-axis, then the ***x-extent*** *of an SOP is the difference between the highest x-coordinate value of a point in the SOP and the lowest x-coordinate value of a point in the SOP.*

Definition 5.13. *If an SOP is plotted in a Cartesian coordinate system such that each edge of the SOP is parallel to the x-axis, y-axis, or z-axis, then the* ***y-extent*** *of an SOP is the difference between the highest y-coordinate value of a point in the SOP and the lowest y-coordinate value of a point in the SOP.*

Definition 5.14. *If an SOP is plotted in a Cartesian coordinate system such that each edge of the SOP is parallel to the x-axis, y-axis, or z-axis, then the* ***z-extent*** *of an SOP is the difference between the highest z-coordinate value of a point in the SOP and the lowest z-coordinate value of a point in the SOP.*

Definition 5.15. *The **diagonal extent** of an SOP is the largest distance between two vertices of the SOP.*

Definition 5.16. *The **3D compactness** of an SOP with volume V, x-extent X, y-extent Y, z-extent Z, and diagonal extent D is given by the relation*

$$Q_3 \equiv \alpha_3 \beta_3 \tag{5.4}$$

where

$$\alpha_3 = \frac{V}{\max\{X, Y, Z\}^3} \tag{5.5}$$

$$\beta_3 = \frac{\alpha_3}{F(D_0)} \tag{5.6}$$

$$D_0 = \frac{D}{\max\{X, Y, Z\}} = \frac{1}{\cos\psi} \tag{5.7}$$

$$F(D_0) = \int_{-0.5}^{0.5}\int_{-0.5}^{0.5}\int_{-0.5}^{0.5} G(x, y, z)\, dx\, dy\, dz \tag{5.8}$$

$$G(x, y, z) = \begin{cases} 0, & x^2 + y^2 + z^2 > 0.25D_0^2 \\ 1, & x^2 + y^2 + z^2 \leq 0.25D_0^2 \end{cases} \tag{5.9}$$

This definition of compactness Q_3 is designed to consider two factors. The first factor α_3 is the ratio between the volume of the SOP and the volume of the smallest cube that can encompass the SOP. This factor serves as a bias against SOPs with a high value for $\max\{X, Y, Z\}/\min\{X, Y, Z\}$. The second factor β_3 is the ratio between α_3 and the maximum possible α_3 that a 3D shape with diagonal extent D inside a cube with side length $\max\{X, Y, Z\}$ can have. This maximum possible α_3 is denoted by $F(D_0)$, where D_0 is the diagonal extent of an SOP scaled to fit inside a cube with side length 1. $F(D_0)$ is then the maximum area that can be covered by a shape with diagonal extent D_0 inside a cube of side length 1. This can be found by calculating the area of the intersection between a cube with side length 1 and a sphere with diameter D_0 concentric with the cube. This factor serves as a bias against SOPs that are longer in one diagonal direction over another

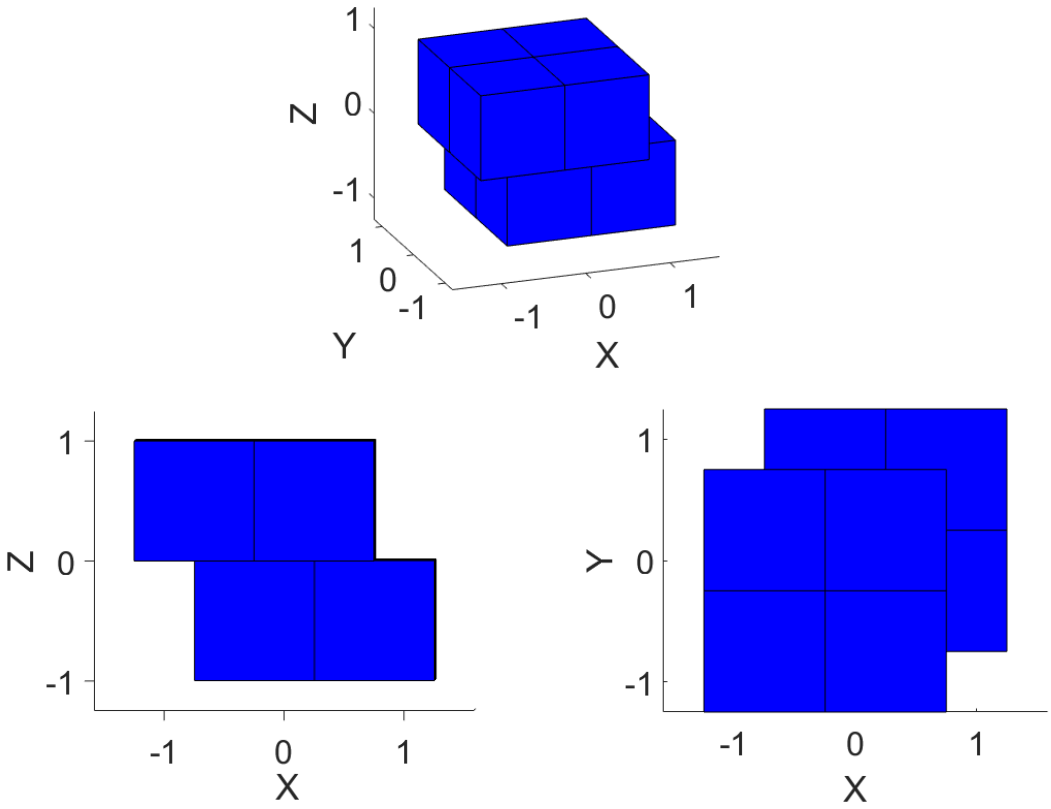


Figure 23: Three viewpoints of an l_1 cell.

diagonal direction. Thus, $Q_3 \leq 1$, and $Q_3 = 1$ iff the SOP is a cube.

The goal now is to maximize Q_3. If each parent cell has eight child cells, then the volume of a cell is $8^n s^3$, where s is the side length of a cube at the finest partition, and n is the level number with level 0 being the finest level. There are multiple ways to form an l_1 cell with eight contiguous child cells. An exhaustive search gives the cell shown in Figure 23 as the solution.

The next goal is to determine the 16 neighbors identical to the surrounded l_1 cell that maximizes r and have the cell and its neighbors be able to be partitioned into cubes arranged in the Λ_3 lattice. In 2D, there were seven different possibilities. In 3D, there are many more possibilities, making an exhaustive search impractical, forgoing the ability to prove clearly that a set of 16 neighbors maximizes r. Instead, first note that viewing the l_1 cell in Figure

23 toward the x-z plane and the y-z plane makes the l_1 cell in 3D look like the l_1 cell in 2D. We know from the 2D case what positions the neighbors must be centered at to get the maximum $r = \sqrt{65}/4$ (assuming the 2D l_0 cell has side length 1). We can extend these 2D positions into 3D positions so that some of the 16 neighbors must be centered at the following positions to get at least a high $r = \sqrt{65}/4$:

$$(x_1, -2, 0), (x_2, -1, -2), (x_3, 1, -2), (x_4, 2, 0), (x_5, 1, 2), (x_6, -1, 2),$$
$$(-2, y_1, 0), (-1, y_2, -2), (1, y_3, -2), (2, y_4, 0), (1, y_5, 2), (-1, y_6, 2).$$

Note that these points are not necessarily mutually exclusive from each other. One solution is the following:

$$(0, -2, 0), (-1, -1, -2), (-1, 1, -2), (0, 2, 0), (-1, 1, 2), (-1, -1, 2),$$
$$(-2, 0, 0), (-1, -1, -2), (1, -1, -2), (2, 0, 0), (1, -1, 2), (-1, -1, 2),$$

which gives 10 out of 16 neighbor centers (two are repeated above). The remaining six to surround the l_1 cell are

$$(1, 1, -2), (-2, -2, 0), (-2, 2, 0), (2, -2, 0), (2, 2, 0), (1, 1, 2).$$

Viewing the union of the surrounded cell and its neighbors toward the x-z plane and the y-z plane would give either the cell as shown in Figure 14 or a reflection of the cell. Note that these 16 points are a subset of a Λ_3 lattice.

There is now a process for generating 3D cells similar to the process for generating 2D cells. The first step is to maximize Q_3 of an l_n cell while maintaining a partition of the problem region in l_{n-1} cells. In the base case, Q_3 is maximized when the cell is a cube. The second step is to arrange the l_n cells in an Λ_3 lattice to get a high value for the shortest distance r between the center of an l_n cell with a boundary point of the union of the cell and

its neighbors. This arrangement sets up the basis for determining the l_{n+1} cell. In the final iteration when the maximum level is reached, only maximizing Q_3 is necessary because the resulting cell would be the root cell and the root cell has no neighbors. Figure 24 shows the first four l_n cells.

5.2 Implementation

The new 3D partition proposed requires a separate piece of code to determine the boundaries of FMM cells at different levels. Fortunately, the proposed 3D partition is a simple extension of the proposed 2D partition. GENERATECELLS was presented as an algorithm for producing 2D l_n cells, each represented as a set of vertices ordered in counterclockwise order along the boundary. Suppose these vertices now represent lines, such as a vertex $(a, b) \in \mathbb{R}^2$ now representing a line $(x, a, b) \in \mathbb{R}^3$, where x is a free variable. Then, each 3D l_n cell can be defined by a set of vertices formed as the intersection points of two sets of lines. Consider an l_1 cell for example. GENERATECELLS outputs the following vertices for a 2D l_1 cell:

$$(-1.25, 0), (-0.75, 0), (-0.75, -1), (1.25, -1), (1.25, 0), (0.75, 0), (0.75, 1), (-1.25, 1).$$

Suppose we create two sets of lines in $\mathbb{R}^3$ from the above vertices in the following way:

$$(x, -1.25, 0), (x, -0.75, 0), (x, -0.75, -1), (x, 1.25, -1),$$
$$(x, 1.25, 0), (x, 0.75, 0), (x, 0.75, 1), (x, -1.25, 1)$$

$$(-1.25, y, 0), (-0.75, y, 0), (-0.75, y, -1), (1.25, y, -1),$$
$$(1.25, y, 0), (0.75, y, 0), (0.75, y, 1), (-1.25, y, 1).$$

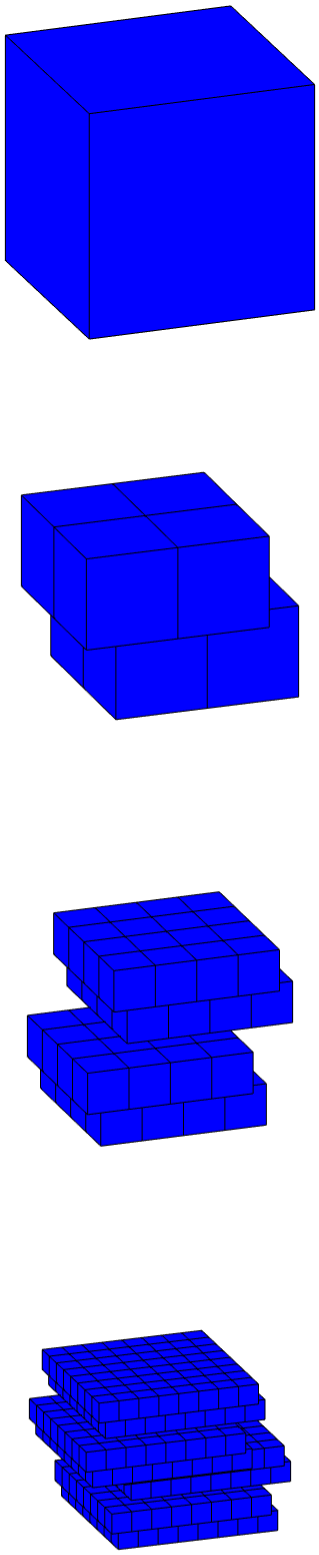


Figure 24: 3D l_n cells for $n \leq 3$.

The intersection points of these two sets of lines form the vertices of the 3D l_1 cell, as shown in Figure 23. Thus, there is no need for a new algorithm. The 3D l_n cells can be created by having a different interpretation of the output of GENERATECELLS.

The next step is to bin points into cells. The process in 3D is similar to the process in 2D. The only major difference is that assigning a point to a 3D l_n cell involves using the point's z-coordinate to determine the vertical planes to compare with the point's x- and y-coordinates. A new algorithm that assigns points from a parent cell to child cells in the proposed 3D partition is presented in Algorithm 3, BINTO3DCELLS.

The solutions to the remaining issues of determining the number the levels, defining what a cell's neighbors are, and defining what cells are members of a cell's interaction list in 3D are analogous to the solutions in 2D. Setting the number of levels at $\lceil \log_8(N/s)+1 \rceil$, where N is the number of points in the problem region and s is the maximum number of points in each leaf cell not in the finest level of cells, for uniformly distributed points and $\lceil \log_8(N/s)+2 \rceil$ or so for nonuniformly distributed points is generally sufficient. Also, there are no substantial algorithmic changes needed for defining neighbors as either explicit lists just need the entries changed or the nature of the cells' positions is enough to determine what each cell's neighbors in the proposed 3D solution are. Because of the definition of the interaction list, interaction lists change automatically as neighbors change.

5.3 Experimental Setup

An open-source 3D FMM code created by Yokota called exaFMM [8] is used as a reference point for 3D FMM performance evaluation. This FMM implementation is compared with a modified version that uses the proposed 3D partition.

Two kernel functions are tested because each kernel function in exaFMM uses different series expansions [8]. The two kernel functions are $\mathbf{r}^{-1}$, where $\mathbf{r}$ is the Euclidean distance between two points, and $e^{jk\mathbf{r}}/\mathbf{r}$, where k is some constant. Each kernel function is used with four point distributions. The first distribution is uniformly distributed points in a cube. The second distribution is uniformly distributed points in a sphere. The third distribution

Algorithm 3 – BINTO3DCELLS An algorithm for assigning points in 3D from a parent l_n cell to child l_{n-1} cells

Input: level number n,
array P of vertices of a 2D l_{n-1} cell centered at $(0, 0)$ in counterclockwise order,
array X of N points in the l_n cell,
center (x_c, y_c, z_c) of the l_n cell,
z-extent Z_n of the l_n cell,
z-extent Z_0 of the l_0 cell

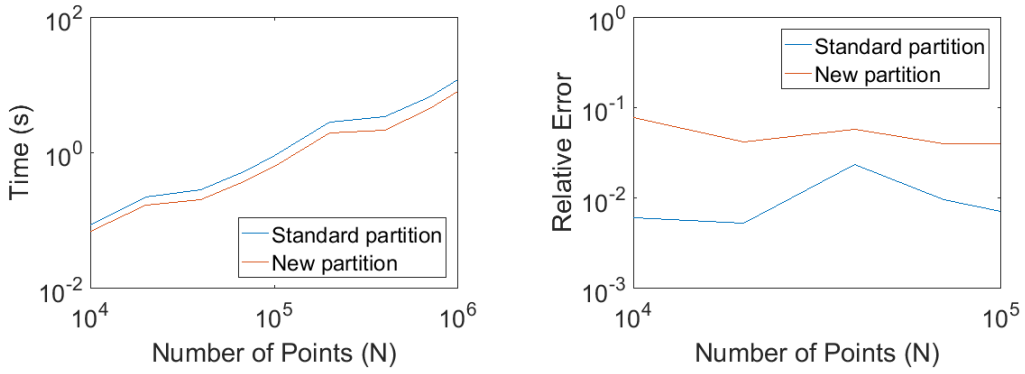
Output: an array of points for each child cell

$P^+ := \{(x, y) \in P \mid x > 0\}$
$(x_1, y_1, z_1) := (x_c, y_c, z_c) + ((-1.5 + n \mod 2)Z_n/4, (-1.5 + n \mod 2)Z_n/4, -Z_n/4)$
$(x_2, y_2, z_2) := (x_c, y_c, z_c) + ((0.5 + n \mod 2)Z_n/4, (-1.5 + n \mod 2)Z_n/4, -Z_n/4)$
$(x_3, y_3, z_3) := (x_c, y_c, z_c) + ((-1.5 + n \mod 2)Z_n/4, (0.5 + n \mod 2)Z_n/4, -Z_n/4)$
$(x_4, y_4, z_4) := (x_c, y_c, z_c) + ((0.5 + n \mod 2)Z_n/4, (0.5 + n \mod 2)Z_n/4, -Z_n/4)$
$(x_5, y_5, z_5) := (x_c, y_c, z_c) + ((-0.5 - n \mod 2)Z_n/4, (-0.5 - n \mod 2)Z_n/4, Z_n/4)$
$(x_6, y_6, z_6) := (x_c, y_c, z_c) + ((1.5 - n \mod 2)Z_n/4, (-0.5 - n \mod 2)Z_n/4, Z_n/4)$
$(x_7, y_7, z_7) := (x_c, y_c, z_c) + ((-0.5 - n \mod 2)Z_n/4, (1.5 - n \mod 2)Z_n/4, Z_n/4)$
$(x_8, y_8, z_8) := (x_c, y_c, z_c) + ((1.5 - n \mod 2)Z_n/4, (1.5 - n \mod 2)Z_n/4, Z_n/4)$
$X_1 := \emptyset, X_2 := \emptyset, X_3 := \emptyset, X_4 := \emptyset, X_5 := \emptyset, X_6 := \emptyset, X_7 := \emptyset, X_8 := \emptyset$
for $i := 0, 1, ..., N - 1$ **do**
    $(x, y, z) := X[i]$
    if $z < z_c$ **then**
        index $:= 2\lfloor z - z_1 + Z_n/4 \rfloor / Z_0$
        $(x', z') := P^+[\text{index}] + (x_1, z_1)$, $(y', z') := P^+[\text{index}] + (y_1, z_1)$
        $j := 2(y > y') + (x > x')$ *// true = 1, false = 0*
        $X_j := X_j \cup (x, y, z)$
    else
        $(x', y', z') := (0, 0, 0)$
        if $z - z_5 = Z_n/4$ **then**
            $(x', z') := P^+[Z_n/Z_0 - 1] + (x_5, z_5)$, $(y', z') := P^+[Z_n/Z_0 - 1] + (y_5, z_5)$
        else
            index $:= 2\lfloor z - z_5 + Z_n/4 \rfloor / Z_0$
            $(x', z') := P^+[\text{index}] + (x_5, z_5)$, $(y', z') := P^+[\text{index}] + (y_5, z_5)$
        end if
        $j := 4 + 2(y > y') + (x > x')$
        $X_j := X_j \cup (x, y, z)$
    end if
end for
return $\{X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8\}$

is uniformly distributed points on a spherical shell. The fourth distribution is Gaussian distributed points in a sphere, where the mean is at the center of the sphere, and the x, y, and z values are each Gaussian random variables with standard deviation 0.194 times the diameter d of the sphere. These standard deviation values were chosen because there would be a 99% chance that $-\frac{d}{2} \leq x, y, z \leq \frac{d}{2}$. For measuring computation time, the number of points in the problem region ranges from 10^4 to 10^6. This range of values is consistent with the range used in prior works [8, 15, 18]. For measuring error, the number of points ranges from 10^4 to 10^5. The number of points for error evaluation does not go beyond 10^5 because error calculation has complexity $O(N^2)$. For each value used for N, an average value from 10 Monte Carlo runs is used to measure computation time and evaluate relative error.

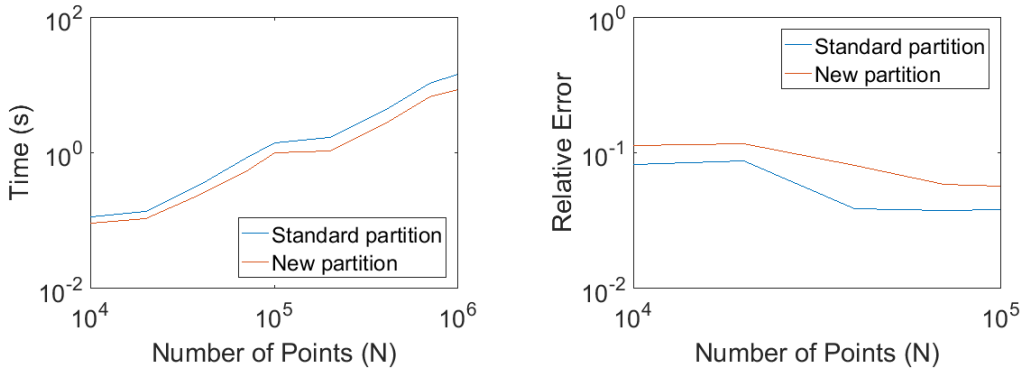
5.4 Results

Figure 25a shows a plot comparing computation time between exaFMM with the standard partition and exaFMM with the new 3D partition for uniformly distributed points in a cube with kernel function $\mathbf{r}^{-1}$. This plot shows the FMM with the new partition performing faster than the FMM with the standard partition by as much as 37%. Figure 25b shows a plot of the corresponding relative error comparison. This plot shows, however, that error is as much as an order of magnitude higher with the new partition in this case.



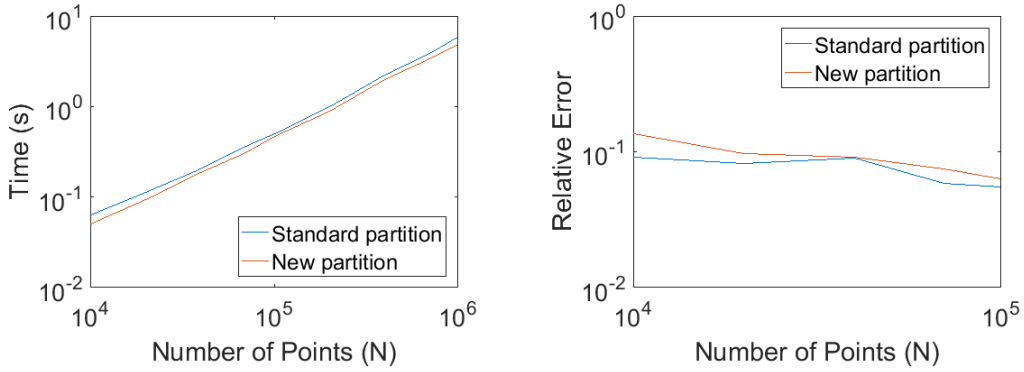
(a) Computation time (b) Relative error

Figure 25: Standard partition vs. new 3D partition for uniformly distributed points in a cube with kernel function $\mathbf{r}^{-1}$.



(a) Computation time (b) Relative error

Figure 26: Standard partition vs. new 3D partition for uniformly distributed points in a sphere with kernel function $\mathbf{r}^{-1}$.

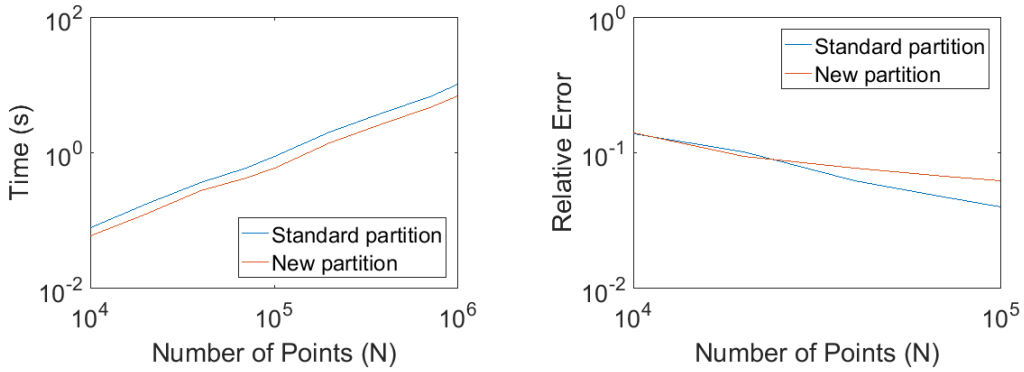


(a) Computation time (b) Relative error

Figure 27: Standard partition vs. new 3D partition for uniformly distributed points on a spherical shell with kernel function $\mathbf{r}^{-1}$.

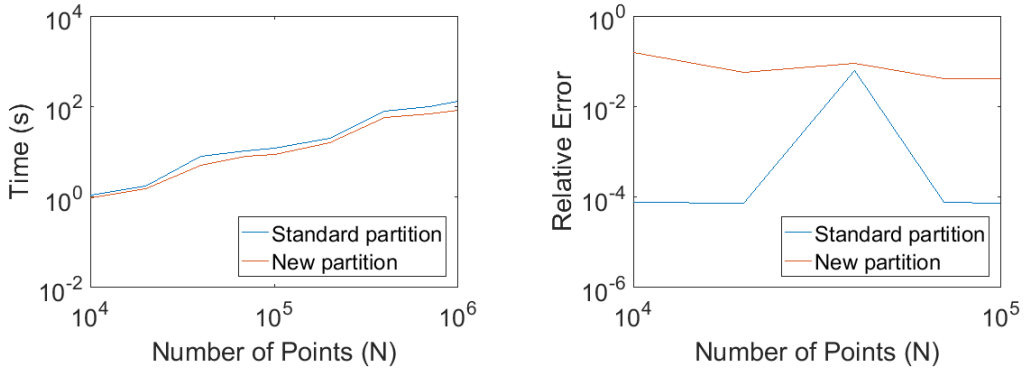
Figure 26 shows plots comparing computation time and relative error for uniformly distributed points in a sphere with kernel function $\mathbf{r}^{-1}$. These figures show that the FMM with the new partition does as well in terms of computation time as with points uniformly distributed in a cube. However, relative error with the new partition is about 50% higher than the error with standard partition, much better compared with the case of points uniformly distributed in a cube.

Figure 27 shows plots comparing computation time and relative error for uniformly distributed points on a spherical shell with kernel function $\mathbf{r}^{-1}$. Here, the new partition provides only modest improvements with only as much as a 20% reduction in computation



(a) Computation time (b) Relative error

Figure 28: Standard partition vs. new 3D partition for Gaussian distributed points in a sphere with kernel function $\mathbf{r}^{-1}$.



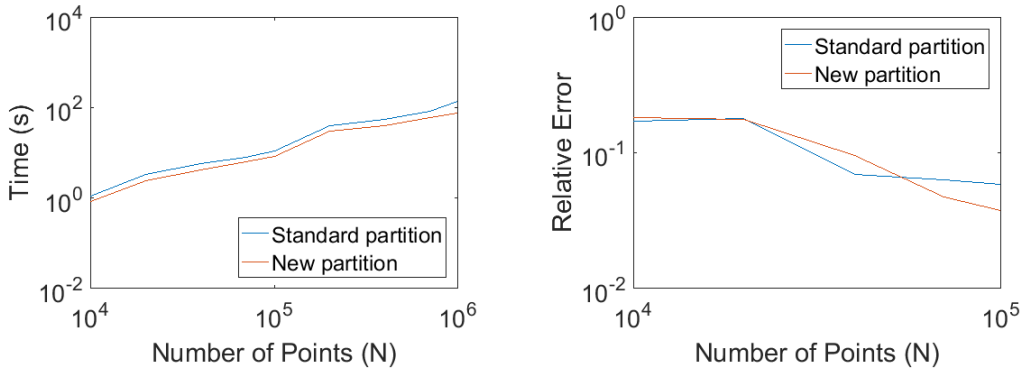
(a) Computation time (b) Relative error

Figure 29: Standard partition vs. new 3D partition for uniformly distributed points in a cube with kernel function $e^{jk\mathbf{r}}/\mathbf{r}$.

time. However, relative error with the new partition is only about 20% higher than the error with the standard partition.

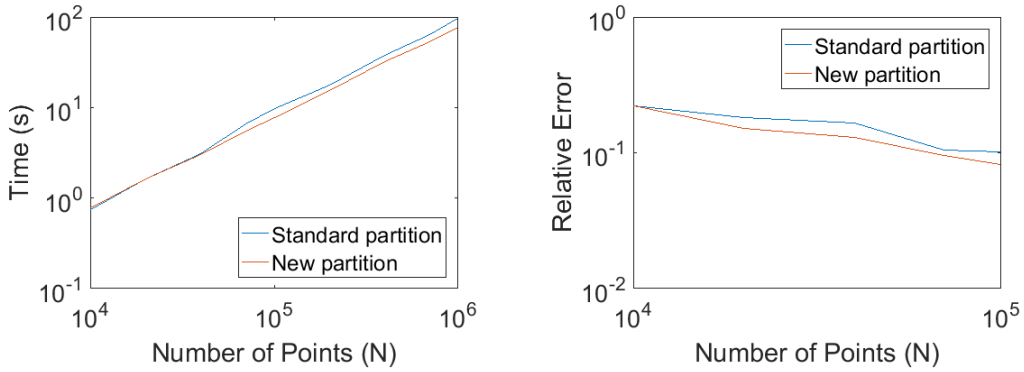
Figure 28 shows plots comparing computation time and relative error for Gaussian distributed points in a sphere with kernel function $\mathbf{r}^{-1}$. Here, the FMM with the new partition is by as much as 33% faster than the FMM with the standard partition. Further, relative error with the new partition is on average 25% higher than the error with the standard partition.

Figure 29 shows plots comparing computation time and relative error for uniformly distributed points in a cube with kernel function $e^{jk\mathbf{r}}/\mathbf{r}$. Here, the FMM with the new



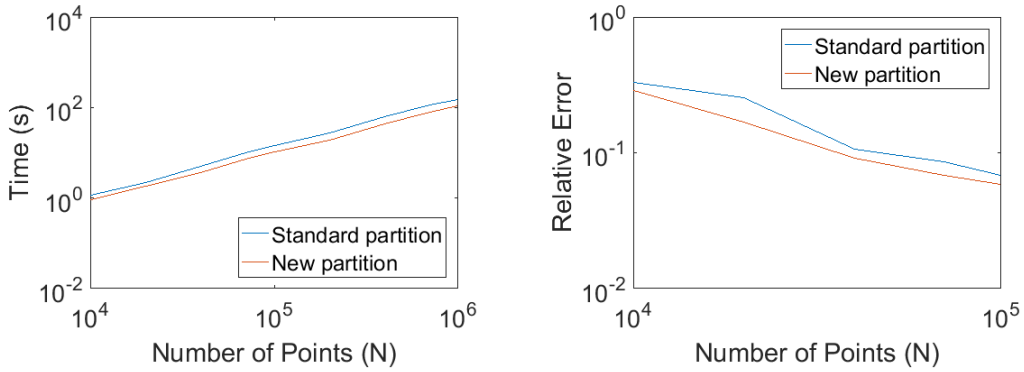
(a) Computation time (b) Relative error

Figure 30: Standard partition vs. new 3D partition for uniformly distributed points in a sphere with kernel function $e^{jk\mathbf{r}}/\mathbf{r}$.



(a) Computation time (b) Relative error

Figure 31: Standard partition vs. new 3D partition for uniformly distributed points on a spherical shell with kernel function $e^{jk\mathbf{r}}/\mathbf{r}$.



(a) Computation time (b) Relative error

Figure 32: Standard partition vs. new 3D partition for Gaussian distributed points in a sphere with kernel function $e^{jk\mathbf{r}}/\mathbf{r}$.

partition is as much as 36% faster than the FMM with the standard partition, but the relative is magnitudes of error higher with the new partition than with the standard partition.

Figure 30 shows plots comparing computation time and relative error for uniformly distributed points in a sphere with kernel function $e^{jk\mathbf{r}}/\mathbf{r}$. Here, the FMM with the new partition is as much as 32% faster than the FMM with the standard partition. The FMM with the new partition has about the same relative error on average as the FMM with the standard partition.

Figure 31 shows plots comparing computation time and relative error for uniformly distributed points in a sphere with kernel function $e^{jk\mathbf{r}}/\mathbf{r}$. Here, the FMM with the new partition is as much as 20% faster than the FMM with the standard partition. The relative error is about 13% lower with the new partition than the error with the standard partition.

Figure 32 shows plots comparing computation time and relative error for uniformly distributed points in a sphere with kernel function $e^{jk\mathbf{r}}/\mathbf{r}$. Here, the FMM with the new partition is as much as 32% faster than the FMM with the standard partition. The relative error is about 23% lower with the new partition than the error with the standard partition.

We now see that the modified partition in the 3D case results in much more significant improvement in many cases than in the 2D case.

CHAPTER 6

CONCLUSION

New and efficient partitions in 2D and 3D were presented in this thesis. These partitions were designed based on the premise that reducing the number of neighbors each cell in a partition has results in reducing the computation time.

6.1 Summary of Results

In 2D FMM, we saw that while computation time was decreased with the new 2D partition, the reduction was not significant. In addition, the new 2D partition resulted in higher error. In 3D FMM, however, we saw that computation time decreased by as much as 37% with the new 3D partition. In most cases, the new 3D partition resulted in little to no additional error. In some cases, the new 3D partition produced smaller error than the standard octree structure did. The only situation where the new 3D partition performed considerably worse than the standard octree structure was the point distribution with sources uniformly distributed in a cube.

6.2 Suggestions for Future Work

Although this thesis focused only on using the FMM in the context of N-body problems, the FMM and the proposed solutions presented can be used in other applications. For example, the FMM can be used to reinterpolate nonuniform samples of a signal into uniform signals [22]. In addition, the FMM can be used for problems where source and target points are not the same [3]. How well the partitions presented will work in other applications remain subject for future research.

While the partitions designed were based on the optimal lattice solutions of the covering problem in 2D and 3D, it is still not known definitively whether other cell arrangements could provide better results. This, too, is also subject for future research.

Finally, the number of child cells each parent cell can be divided into was kept the

same as the number in the quadtree in 2D and the number in octree structures. Although there is no clear advantage for changing this number, that does not mean a partition more efficient than the partitions presented where the number of child cells is different does not exist. Whether changing the number of child cells can be proven to be suboptimal or if a systemic means of creating more efficient partitions with different numbers of child cells exists is still an open question.

REFERENCES

[1] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of Computational Physics*, vol. 4, no. 2, pp. 325–348, 1987.

[2] B. A. Cipra, "The best of the 20th century: Editors name top 10 algorithms," *SIAM News*, vol. 33, no. 4, 2000.

[3] L. Ying, "A pedestrian introduction to fast multipole methods," *Science China Mathematics*, vol. 55, no. 5, pp. 1043–1051, 2012.

[4] K. Kudin and G. Scuseria, "A fast multipole method for periodic systems with arbitrary unit cell geometries," *Chemical Physics Letters*, vol. 283, no. 1-2, pp. 61–68, 1998.

[5] R. Anderson, "Tree data structures for N-body simulation," *SIAM J. Comput.*, vol. 28, no. 6, pp. 1923–1940, 1999.

[6] Y. Marzouk and A. Ghoniem, "K-means clustering for optimal partitioning and dynamic load balancing of parallel hierarchical N-body simulations," *Journal of Computational Physics*, vol. 207, no. 2, pp. 493–528, 2005.

[7] L. Ying, G. Biros, and D. Zorin, "A kernel-independent adaptive fast multipole algorithm in two and three dimensions," *Journal of Computational Physics*, vol. 196, no. 2, pp. 591–626, 2004.

[8] R. Yokota, "An FMM based on dual tree transversal for many-core architectures," *Journal of Algorithms & Computational Technology*, vol. 7, no. 3, pp. 301–324, 2013.

[9] J. Fraleigh, *A First Course in Abstract Algebra*. Reading, MA: Addison-Wesley Pub. Co., 1982.

[10] H. Coxeter, "Discrete groups generated by reflections," *The Annals of Mathematics*, vol. 35, no. 3, pp. 588–621, 1934.

[11] A. Bjorner and F. Branti, *Combinatorics of Coxeter Groups*. New York, NY: Springer, 2005.

[12] M. Davis, *The Geometry and Topology of Coxeter Groups*. Princeton, NJ: Princeton University Press, 2008.

[13] C. P. Bruter, *Mathematics and Art*. Berlin: Springer, 2005.

[14] J. Conway and N. Sloane, *Sphere Packings, Lattices, and Groups*. New York, NY: Springer-Verlag, 1988.

[15] W. Fong and E. Darve, "The black-box fast multipole method," *Journal of Computational Physics*, vol. 228, no. 23, pp. 8712–8725, 2009.

[16] J. R. Munkres, *Topology*. Upper Saddle River, NJ: Prentice Hall, Inc., 2000.

[17] P. Bourke, "Polygon area and centroid," 1988.

[18] H. Cheng, L. Greengard, and V. Rokhlin, "A fast adaptive multipole algorithm in three dimensions," *Journal of Computational Physics*, vol. 155, no. 2, pp. 468–498, 1999.

[19] J. C. Mason and D. C. Handscomb, *Chebyshev Polynomials*. Boca Raton, FL: Chapman & Hall/CRC, 2003.

[20] R. Nurnberg, "Calculating the volume and centroid of a polyhedron in 3d," 2013.

[21] E. W. Nelson, C. L. Best, and W. G. McLean, *Schaum's Outline of Theory and Problems of Engineering Mechanics: Statics and Dynamics*. New York, NY: McGraw-Hill, 1998.

[22] C. K. Turnes, *Efficient solutions to Toeplitz-structured linear systems for signal processing*. PhD thesis, School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 2014.